

UNIVERSITATEA “TRANSILVANIA” DIN BRAȘOV

Laurențiu-Mihail IVANOVICI

Procesarea Imaginilor



Îndrumar de laborator

2006

© 2003 EDITURA UNIVERSITĂȚII *TRANSILVANIA* BRAȘOV

Adresa: 500030 Brasov,
B-dul Eroilor, Nr. 9
Tel/Fax: 0268 - 47 53 48
E-mail: editura@unitbv.ro



Tipărit la:
Tipografia Universității “Transilvania” din Brașov
B-dul Eroilor 9
tel/fax: 0268 47 53 48

Toate drepturile rezervate

Editură acreditată de CNCSIS
Adresa nr. 1615 din 29 mai 2002

Referenți: Prof. dr. ing. Iuliu Szekely
Ș. l. dr. ing. Angel Cațaron

Descrierea CIP a Bibliotecii Naționale a României
IVANOVICI, LAURENȚIU MIHAIL

Procesarea imaginilor: îndrumar de laborator /
Laurențiu-Mihail Ivanovici. - Brașov: Editura Universității
“Transilvania”, 2006

Bibliogr.

ISBN (10) 973-635-674-4; ISBN (13) 978-973-635-674-2

004.932

Cuprins

Cuprins	i
Mulțumiri	v
Cuvânt înainte	ix
1 Noțiuni introductive de Qt	1
1.1 Clasa <code>QApplication</code>	1
1.2 Clasa <code>QImage</code>	1
1.2.1 Manipularea imaginilor	2
1.2.2 Atribute ale imaginii	2
1.2.3 Manipularea pixelilor	3
1.2.4 Formate de imagine	4
2 Noțiuni introductive	5
2.1 Percepția imaginilor	5
2.1.1 Structura ochiului uman. Formarea imaginii	5
2.1.2 Lumina. Luminanța. Strălucirea	6
2.2 Modelul matematic al imaginii	7
2.2.1 Modelul continuu al imaginii	7
2.2.2 Modelul discret al imaginii	8
2.3 Eșantionarea imaginilor	8
2.4 Cuantizarea imaginilor	9
2.5 Imaginile digitale	9
3 Îmbunătățirea imaginilor	15
3.1 Operațiile punctuale	15
3.2 Accentuarea contrastului	16
3.3 Întinderea maximă a contrastului	17
3.4 Binarizarea imaginilor	18
3.5 Negativarea imaginilor	19
3.6 Decuparea imaginilor	20

4	Egalizarea histogramei	25
4.1	Histograma unei imagini	25
4.2	Egalizarea histogramei	26
4.2.1	Algoritmul de egalizare a histogramei	26
4.3	Observații	27
5	Transformări geometrice de bază	31
5.1	Translația	31
5.2	rotația	31
5.3	Oglindirea	33
6	Zgomotul în imagini	37
6.1	Zgomotul cu distribuție uniformă	38
6.2	Zgomotul cu distribuție gaussiană	39
6.3	Zgomotul de tip “sare și piper”	41
6.4	Alte tipuri de zgomot	41
7	Filtrarea imaginilor	47
7.1	Filtrarea liniară a imaginilor	47
7.1.1	Filtrele de netezire	48
7.1.2	Filtrele trece-sus	51
7.2	Filtrarea neliniară a imaginilor	52
7.2.1	Filtrele de ordine	53
8	Transformări unitare	59
8.1	Transformări unitare bidimensionale	60
8.1.1	Proprietățile transformărilor unitare	62
8.2	Transformata Fourier discretă	62
8.2.1	Transformata Fourier unidimensională	62
8.2.2	Transformarea Fourier bidimensională	63
8.3	Transformata cosinus discretă	64
8.4	Transformata sinus discretă	65
9	Compresia imaginilor	69
9.1	Clasificarea metodelor de compresie	69
9.2	Algoritmul Huffman	70
9.3	Algoritmul RLE	73
9.3.1	Algoritmul RLE pentru imagini binare	73
9.3.2	Algoritmul RLE pentru imagini în tonuri de gri	73
9.4	Compresia cu transformate	74
10	Segmentarea imaginilor	81
10.1	Segmentarea orientată pe regiuni	81
10.1.1	Prăguirea histogramei	82
10.1.2	Segmentarea prin creștere de regiuni	83

10.2	Segmentarea orientată pe contururi	83
10.2.1	Tehnicile de gradient	84
10.2.2	Operatorii compas	86
10.2.3	Identificarea trecerilor prin zero ale celei de-a doua derivate	87
Lista figurilor		90
Bibliografie		92

Dedicație

Doresc să dedic această carte doamnei Delia Gârbacea.

Mulțumiri

Țin să mulțumesc domnului profesor Iuliu Szekely, conducătorul catedrei de Electronică și Calculatoare, din cadrul Universității “Transilvania” din Brașov, pentru încrederea pe care mi-a acordat-o prin încredințarea orelor de laborator de procesarea imaginilor.

Aș vrea să mulțumesc domnului profesor Vasile Buzuloiu, conducătorul Laboratorului de Analiza și Prelucrarea Imaginilor (LAP), din cadrul Universității “POLITEHNICA” din București, pentru sprijinul și îndrumarea acordate în formarea mea profesională.

Vreau să mulțumesc colegilor din cadrul LAPI, conferențiar Mihai Ciuc și conferențiar Constantin Vertan, pentru discuțiile purtate de-a lungul anilor, în special cele care au vizat lucrările de laborator de procesarea imaginilor.

Nu în ultimul rând, vreau să mulțumesc studenților care, cu sugestii sau comentarii, au contribuit la îmbunătățirea conținutului acestui îndrumar de laborator.

Cuvânt înainte

Procesarea imaginilor este un domeniu relativ recent, care evoluează rapid. Aplicațiile sale se întâlnesc pretutindeni: în medicină, armată, industrie, artă sau acolo unde informația din mediul înconjurător este reprezentată sub formă de imagini. Principala aplicație o reprezintă îmbunătățirea informației conținute de imagini în vederea interpretării de către un subiect uman sau pentru vederea artificială a roboților.

Odată cu răspândirea aparatelor de fotografiat digitale, procesarea imaginilor nu mai este domeniul exclusiv al oamenilor de știință sau al inginerilor. Aplicațiile de procesare a imaginilor au pătruns în casa oricărui fotograf amator, fiind utilizate pentru retușarea fotografiilor digitale.

Îndrumarul de față se adresează în special studenților de la facultățile tehnice, care urmează cursul de Procesarea imaginilor, dar și celor care doresc să își însușească algoritmi fundamentali de procesare a imaginilor. Lucrările de laborator prezentate abordează operațiile punctuale, de vecinătate și cele integrale, de la accentuarea contrastului până la transformări unitare. Ultimul capitol, referitor la segmentarea imaginilor, reprezintă o incursiune în domeniul analizei imaginilor.

Scopul lucrărilor de laborator este acela de a prezenta noțiunile fundamentale de procesare a imaginilor alb-negru și de a exemplifica folosind limbajul C++ modalități de implementare a algoritmilor de procesare prezentați. Pentru o înțelegere cât mai bună a soluțiilor propuse, cititorul trebuie să fie familiarizat cu limbajele de programare C și C++.

Lucrarea 1

Noțiuni introductive de Qt

Qt¹ este un mediu de dezvoltare care include biblioteci de clase C++ și unelte de dezvoltare de aplicații pentru diverse sisteme de operare: Microsoft Windows, MAC OS sau Linux.

Bibliotecile **Qt** cuprind peste 400 de clase C++, care încapsulează infrastructura necesară pentru dezvoltarea de aplicații. **Qt API**² include clase pentru realizarea de interfețe grafice utilizator, pentru programarea în rețea, baze de date sau integrare OpenGL³.

1.1 Clasa `QApplication`

Clasa `QApplication` controlează și gestionează interfața grafică a unei aplicații. Conține bucla principală de evenimente a aplicației, în care sunt procesate evenimentele provenind de la sistemul de ferestre. De asemenea gestionează configurările aplicației, fazele de inițializare și terminare a aplicației.

Orice aplicație **Qt** este un obiect `QApplication` indiferent de numărul de ferestre ale aplicației. Obiectul este accesibil prin apelarea funcției membre `instance()`, care returnează un pointer către acest obiect.

1.2 Clasa `QImage`

Clasa `QImage` pune la dispoziție o reprezentare independentă de platformă a unei imagini, care permite accesul direct la valorile pixelilor. **Qt** oferă patru clase pentru manipularea imaginilor: `QImage`, `QPixmap`, `QBitmap` și `QPicture`.

Clasa `QImage` este proiectată și optimizată pentru operații de intrare/ieșire și pentru acces direct la pixelii imaginii, pe când clasa `QPixmap` este proiectată și optimizată pentru vizualizarea imaginilor pe ecran. Clasa `QBitmap`

¹Trolltech, <http://www.trolltech.com>

²Application Program Interface.

³Limbaj de grafică 3D dezvoltat de firma Silicon Graphics.

moștenește clasa `Pixmap` și este folosită pentru imagini binare, iar clasa `QPicture` este un “paint device”. În continuare ne vom concentra atenția asupra clasei `QImage`.

Clasa `QImage` poate manipula o serie de formate de imagine, care includ fie imagini monocrome, fie reprezentate pe 8 biți sau 32 biți. Clasa pune la dispoziție o colecție de funcții care pot fi folosite pentru obținerea de informații despre imagine sau care permit anumite transformări ale imaginii.

1.2.1 Manipularea imaginilor

Clasa `QImage` oferă câteva moduri de a încărca o imagine dintr-un fișier: la instanțierea obiectului `QImage` sau folosind funcțiile `loadFromData()` sau `load()`, care vor fi apelate după crearea obiectului `QImage`. Pentru a salva un obiect de tip `QImage` se folosește funcția `save()`.

Următorul exemplu ilustrează folosirea funcției `load()`:

```
QImage image;
QString filename = QFileDialog :: getOpenFileName();
image.load( filename, 0 );
```

Lista completă a formatelor de fișiere cunoscute este disponibilă prin intermediul a două metode: `QImageReader::supportedImageFormats()` și `QImageWriter::supportedImageFormats()`. Implicit, **Qt** suportă următoarele formate de fișiere:

Format	Descriere	Operații
BMP	Windows Bitmap	Read/Write
GIF	Graphic Interchange Format (optional)	Read
JPG	Joint Photographic Experts Group	Read/Write
JPEG	Joint Photographic Experts Group	Read/Write
PNG	Portable Network Graphics	Read/Write
PBM	Portable Bitmap	Read
PGM	Portable Graymap	Read
PPM	Portable Pixmap	Read/Write
XBM	X11 Bitmap	Read/Write
XPM	X11 Pixmap	Read/Write

Tabelul 1.1: Formate de fișiere imagine suportate de **Qt**.

1.2.2 Atribute ale imaginii

`QImage` oferă o colecție de funcții pentru obținerea de informații despre atributele imaginii, cum ar fi geometria imaginii sau informații despre paleta de culori.

Atribute	Funcții
Geometrie	Funcțiile <code>size()</code> , <code>width()</code> , <code>height()</code> , <code>dotsPerMeterX()</code> și <code>dotsPerMeterY()</code> oferă informații despre dimensiunile și aspectul imaginii.
Culoare	Culoarea unui pixel poate fi aflată specificând coordonatele pixelului funcției <code>pixel()</code> , care returnează culoarea ca o valoare <code>QRgb</code> . În cazul imaginilor monocrome sau cu nivele de gri, funcțiile <code>numColors()</code> și <code>colorTable</code> oferă informații despre componentele de culoare ale imaginii.
Nivel jos	Funcția <code>depth()</code> returnează numărul de biți pe care este reprezentată valoarea unui pixel: 1 (imagini monocrome), 8 sau 32. Funcțiile <code>format()</code> , <code>bytesPerLine()</code> și <code>numBytes()</code> oferă informații despre modul de stocare a imaginii.

Tabelul 1.2: Funcții de aflare a atributelor unei imagini.

1.2.3 Manipularea pixelilor

În cazul în care culorile pixelilor sunt stocate pe 32 biți, funcția `setPixel` se poate utiliza pentru modificarea valorii unui pixel specificat prin coordonatele sale. Noua sa valoare este un cvadruplu ARGB⁴, fiind unul din argumentele funcției. Pentru a specifica valoarea unui pixel în modelul RGB se folosește funcția `qRgb` care returnează un obiect `QRgb`.

În continuare este prezentat un exemplu de folosire a funcției `setPixel`. Se crează mai întâi un obiect `QImage`, reprezentând o imagine de dimensiune 3 x 3.

```
QImage image(3, 3, QImage::Format_RGB32);
QRgb value;

value = qRgb(189, 149, 39); // 0xffbd9527
image.setPixel(1, 1, value);

value = qRgb(122, 163, 39); // 0xff7aa327
image.setPixel(0, 1, value);
image.setPixel(1, 0, value);
```

⁴Alpha Red Green Blue.

```
value = qRgb(237, 187, 51); // 0xffedba31
image.setPixel(2, 1, value);
```

1.2.4 Formate de imagine

Fiecare pixel stocat într-un obiect de tip `QImage` este reprezentat ca un întreg. Dimensiunea acestui întreg depinde de format. **Qt** suportă imagini monocrome, reprezentate pe 1 bit și imagini reprezentate pe 8 sau 32 biți.

Imaginile monocrome sunt stocate folosind indici de 1 bit într-o tabelă de culoare cu doar două intrări (culori). În funcție de ordinea de stocare a biților, big endian sau little endian, diferențiem două tipuri de imagini monocrome.

Imaginile reprezentate pe 8 biți sunt stocate folosind indici de 8 biți într-o tabelă de culoare, având un byte per pixel. Tabela de culoare este un obiect `QVector<QRgb>`.

Imaginile reprezentate pe 32 biți nu au tabelă de culoare, fiecare pixel conținând o valoare `QRgb`. Cea mai răspândită modalitate de a reprezenta tripletul RGB pe 32 biți este următoarea: `0xffRRGGBB`, în care se folosesc câte 8 biți pentru fiecare componentă.

Formatul unei imagini se poate determina folosind funcția `format()`.

Pentru a converti o imagine într-un nou format se folosește funcția `convertToFormat()`, care acceptă ca argument noul format al imaginii.

Lucrarea 2

Noțiuni introductive de prelucrarea imaginilor

BREVIAR TEORETIC

2.1 Percepția imaginilor

Înțelegerea procesului de percepție a imaginilor de către ochiul uman este foarte importantă pentru dezvoltarea de tehnici de evaluare a calității unui sistem sau algoritm de procesare a imaginilor. Informația la nivel vizual, conținută de către o imagine, reprezintă o distribuție spațială a unei anumite mărimi, cum ar fi de exemplu luminanța obiectelor ce compun respectiva imagine. Informația percepută de ochiul uman poate fi definită de atribute cum ar fi strălucirea, culoarea sau muchiile unui obiect.

2.1.1 Structura ochiului uman. Formarea imaginii

Ochiul uman are o formă aproape sferică, având un diametru, în medie, de aproximativ 2 cm. Este format din mai multe membrane: *corneea* și *sclerotica*, ca înveliș exterior, și *coroida* și *retina* în interior. Corneea este un țesut dur și transparent, pe când sclerotica este o țesut opac. Coroida se află imediat sub sclerotică și conține o rețea de vase de sânge, ce hrănesc ochiul. Este puternic pigmentată pentru a reduce excedentul de lumină ce intră în ochi. Pe cea mai din interior membrană, retina, se formează imaginea, sub influența luminii reflectate de obiectele exterioare ochiului.

Retina conține două tipuri de receptori: *conuri* și *bastonașe*. Conurile, în număr de aproximativ 6-7 milioane, servesc la percepția culorilor. Vederea umană poate percepe detalii foarte fine datorită densității mari a acestor receptori, fiecare con fiind legat la o terminație nervoasă. Bastonașele, în număr de aproximativ 75-150 milioane, servesc vederii crepusculare, în condiții de iluminare slabă. Acestea sunt răspândite pe o arie mare și conec-

tate la doar câteva terminații nervoase, ceea ce are ca efect reducerea considerabilă a percepției detaliilor din imagine.

Cristalinul joacă rolul de lentilă. El conține aproximativ 70% apă și, prin compoziția lui, permite trecerea a doar 8% din spectrul de radiație vizibilă, absorbind în bună măsură și radiațiile infraroșii și ultraviolete.

Imaginea obiectelor se proiectează pe retină, prin cristalin, fiind răsturnată și având dimensiuni mult mai mici, principiu ce stă la baza aparatului de fotografiat (vezi Figura 2.1).

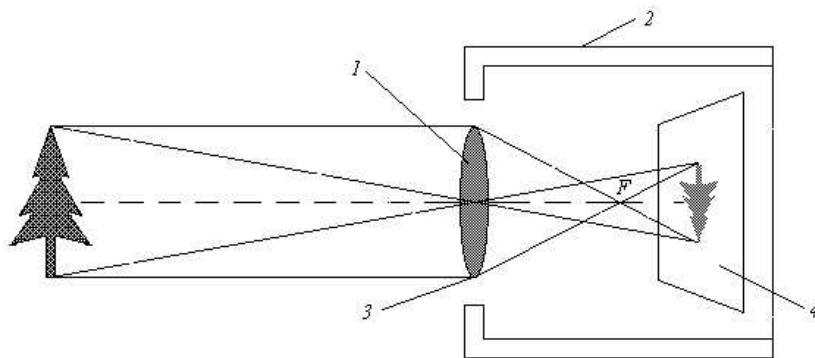


Figura 2.1: Formarea imaginii în aparatul de fotografiat: 1-lentilă, 2-camera obscură, 3-obiectiv, 4-peliculă.

2.1.2 Lumina. Luminanța. Strălucirea

Lumina este o radiație electromagnetică ce stimulează receptorii de la nivelul retinei. Ea se exprimă ca fiind o distribuție $L(\lambda)$ de energie, unde λ este lungimea de undă a radiației, în cazul nostru vizibilă, cu valori între 350 și 780 nm. Lumina percepută de la un obiect se poate scrie matematic astfel:

$$I(\lambda) = \rho(\lambda)L(\lambda) \quad (2.1)$$

unde $\rho(\lambda)$ reprezintă măsura în care un obiect reflectă sau transmite energia luminoasă incidentă, a cărei distribuție este exprimată prin $L(\lambda)$.

Luminanța sau intensitatea luminoasă a unui obiect cu o distribuție spațială a luminii, $I(x, y, \lambda)$, se definește astfel:

$$f(x, y) = \int_0^{\infty} I(x, y, \lambda)V(\lambda)d\lambda \quad (2.2)$$

unde $V(\lambda)$ este *funcția de eficiență luminoasă relativă* a sistemului vizual. Pentru ochiul uman, $V(\lambda)$ este o curbă de tip clopot, a cărei caracteristici depind de la o persoană la alta (vezi Figura 2.2).

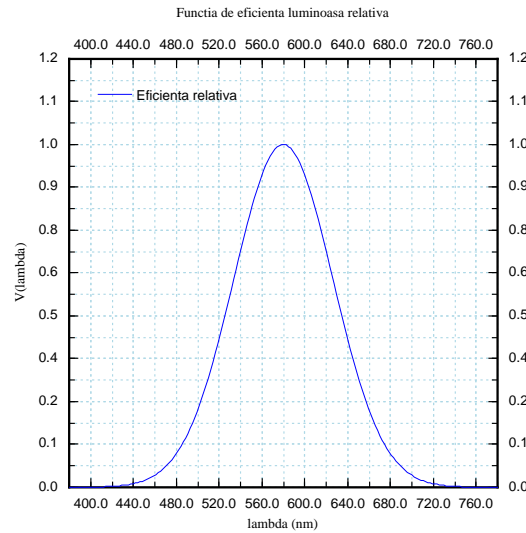


Figura 2.2: Forma tipică a funcției de eficiență luminoasă relativă.

Luminanța unui obiect este independentă de luminanța obiectelor din jur.

Strălucirea unui obiect este luminanța percepută și depinde de luminanța mediului ambiant obiectului. Două obiecte aflate într-un același ambiant pot avea aceeași luminanță, dar străluciri diferite.

2.2 Modelul matematic al imaginii

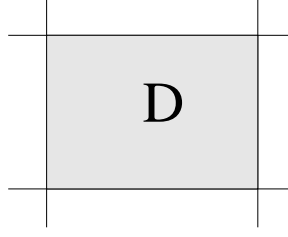
2.2.1 Modelul continuu al imaginii

Matematic imaginile pot fi reprezentate ca o funcție de două variabile, în spațiul $L^2(R^2)$, astfel:

- imaginile în tonuri de gri se pot modela cu: $f(x, y) : R^2 \rightarrow R$, caz în care valorile funcției f reprezintă valorile luminanței obiectelor din imagine, în punctele (x, y) ale spațiului.
- imaginile color se pot modela cu: $f(x, y) : R^2 \rightarrow R^3$, caz în care valorile funcției f reprezintă vectori de 3 componente dintr-un spațiu al culorilor. De exemplu pot fi cele trei componente ale modelului RGB¹.

Spațiul $L^2(R^2)$ poate fi limitat la un domeniu finit D , ca în Figura 2.3.

¹RGB = (engl.) Red Green Blue.

Figura 2.3: Domeniu finit din R^2 .

2.2.2 Modelul discret al imaginii

Acesta este modelul utilizat în practică. Funcția f ia valori discrete, fiind deasemenea definită pe un domeniu de valori discrete, adică:

$$f(k, l) : Z^2 \rightarrow Z^+ \text{ sau } f(k, l) : Z^2 \rightarrow Z^{+3} \quad (2.3)$$

Trecerea de la domeniul continuu la domeniul discret se face prin eșantionare și cuantizare.

2.3 Eșantionarea imaginilor

Pentru a putea prelucra cu ajutorul unui calculator o imagine $f(x, y)$, aceasta trebuie discretizată spațial și în amplitudine. Discretizarea coordonatelor spațiale (x, y) poartă numele de *eșantionare*.

Eșantionarea reprezintă procesul de aproximare a unei imagini continue $f(x, y)$ cu o matrice de dimensiune $M \times N$, astfel:

$$f(x, y) \approx \begin{pmatrix} f(0, 0) & f(0, 1) & \dots & f(0, M-1) \\ f(1, 0) & f(1, 1) & \dots & f(1, M-1) \\ \vdots & \vdots & \dots & \vdots \\ f(N-1, 0) & f(N-1, 1) & \dots & f(N-1, M-1) \end{pmatrix} \quad (2.4)$$

Teorema eșantionării: O imagine $f(x, y)$, având un spectru finit, eșantionată uniform cu o rețea dreptunghiulară de forma celei din Figura 2.4 poate fi refăcută fără eroare din eșantioanele $f(m\delta x, n\delta y)$ cu ajutorul formulei de interpolare:

$$f(x, y) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(m\delta x, n\delta y) \left(\frac{\sin \pi(xu_s - m)}{\pi(xu_s - m)} \right) \left(\frac{\sin \pi(xv_s - n)}{\pi(xv_s - n)} \right) \quad (2.5)$$

unde u_s și v_s reprezintă frecvențele spațiale de eșantionare.

Egalitatea dată de teorema eșantionării este valabilă dacă și numai dacă este respectată condiția Nyquist, și anume:

$$\frac{1}{\delta x} = u_s > 2u_0, \quad \frac{1}{\delta y} = v_s > 2v_0 \quad (2.6)$$

unde u_0 și v_0 reprezintă frecvențele spațiale maxime care apar în imagine.

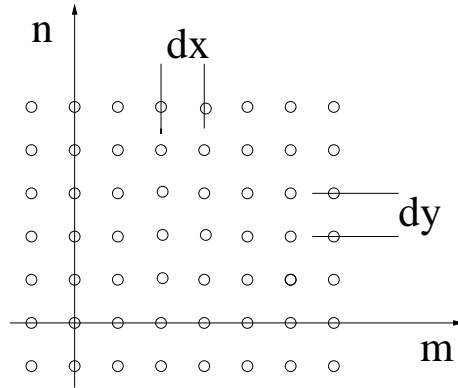


Figura 2.4: Rețea dreptunghiulară de eșantionare.

Cu alte cuvinte, frecvențele spațiale de eșantionare trebuie să fie cel puțin dublul frecvențelor spațiale maxime conținute de imagine.

2.4 Cuantizarea imaginilor

Cuantizarea este procesul de discretizare a valorilor funcției $f(x, y)$. Aceasta se realizează de obicei cu ajutorul unei funcții de tip scară, de forma celei din Figura 2.5.

Astfel, tuturor valorilor lui x dintr-un interval li se atribuie valori discrete k . Cuantizarea este un proces însoțit de zgomot, cunoscut sub numele de *eroare de cuantizare*. Cea mai utilizată metodă de cuantizare este cea uniformă, ceea ce înseamnă că intervalele funcției de cuantizare sunt egale.

2.5 Imaginile digitale

Imaginile astfel discretizate reprezintă structuri bidimensionale de date, denumite imagini digitale. Un element (k, l) al imaginii poartă numele de *pixel*².

²pixel = (engl. picture + element).

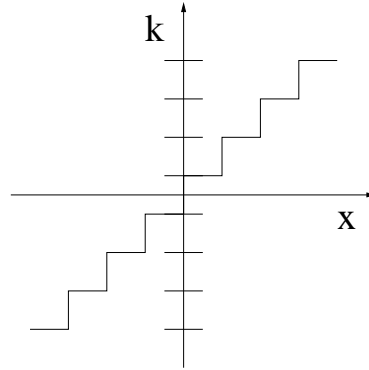


Figura 2.5: Exemplu de funcție de cuantizare.

Imaginile digitale pot fi stocate în memoria sau pe discul unui sistem de procesare și analiză a imaginilor, în vederea vizualizării sau prelucrării ulterioare. Pe disc imaginile sunt stocate sub forma unor fișiere. Fișierele pot fi de mai multe feluri, în funcție de formatul în care păstrează datele ce reprezintă imagini: BMP, JPEG, GIF, TIFF, etc.

Cel mai simplu format de fișier imagine este Windows Bitmap (BMP) al firmei Microsoft. Acesta este formatul în care o imagine digitală este stocată practic fără nici un fel de codare sau pierdere de informație, cu excepția reprezentării binare.

DESFĂȘURAREA LUCRĂRII

În continuare este prezentat codul C++ al unei aplicații care citește o imagine în format BMP și o afișează într-o fereastră pe ecran. Aplicația este dezvoltată utilizând bibliotecile Qt (Linux). Citiți și înțelegeți codul.

Fișierul `aplicatie.h` conține declararea clasei `ImageViewer`.

```
#ifndef APLICATIE_H
#define APLICATIE_H

#include <qwidget.h>
#include <qimage.h>
#include <qpainter.h>
#include <qlabel.h>

class QMenuBar;
class QPopupMenu;

class ImageViewer : public QWidget
```

```

{
    Q_OBJECT
public:
    ImageViewer( QWidget *parent = 0, const char *name = 0,
                int wFlags = 0 );
    ~ImageViewer();
    bool loadImage( const char *fileName );

protected:
    void paintEvent( QPaintEvent * );

private:
    int      conversion_flags;
    int      alloc_context;
    QImage   image;
    QPixmap  pm;
    QMenuBar *menubar;
    QPopupMenu *file;
    QLabel   *status;
    bool      reconvertImage();

private slots:
    void openFile( void );
    void saveFile( void );
};

#endif // APLICATIE_H

```

Fișierul aplicatie.cpp conține definițiile funcțiilor membre clasei ImageViewer.

```

#include "aplicatie.h"
#include <qmenubar.h>
#include <qfiledialog.h>
#include <qmessagebox.h>
#include <qpopupmenu.h>
#include <qpainter.h>
#include <qapplication.h>
#include <qwidget.h>

ImageViewer :: ImageViewer( QWidget *parent, const char *name,
    int wFlags ) : QWidget( parent , name , wFlags ),
    conversion_flags( PreferDither ), filename( 0 )
{
    alloc_context = 0;

```

```

menubar = new QMenuBar( this );
menubar->setSeparator( QMenuBar :: InWindowsStyle );

file = new QPopupMenu();
CHECK_PTR( file );
menubar->insertItem( "&File" , file );
file->insertItem( "Open image ( BMP )", this,
    SLOT( openFile() ), CTRL+Key_O );
file->insertSeparator();
file->insertItem( "Save as ... ( BMP )", this,
    SLOT( saveFile() ), CTRL+Key_S );
file->insertSeparator();
file->insertItem( "Quit", qApp, SLOT(quit()), CTRL+Key_Q );

status = new QLabel( this );
status->setFrameStyle( QFrame::WinPanel | QFrame::Sunken );
status->setFixedHeight( fontMetrics().height() + 4 );

setMouseTracking( TRUE );
}

ImageViewer :: ~ImageViewer()
{
    if ( alloc_context )
        QColor :: destroyAllocContext( alloc_context );

    delete menubar;
    delete file;
    delete help;
    delete status;
}

void ImageViewer :: openFile()
{
    QString newfilename = QFileDialog :: getOpenFileName();

    if ( !newfilename.isEmpty() )
    {
        loadImage( newfilename );
        repaint();
    }
}

void ImageViewer :: saveFile( void )

```



```

{
    QImageIO iio;
    QString save_as_file = QFileDialog :: getSaveFileName();

    if( !save_as_file.isEmpty() )
    {
        iio.setImage( image );
        iio.setFileName( save_as_file );
        iio.setFormat( "BMP" );
        iio.write();
    }
}

bool ImageViewer :: loadImage( const char *fileName )
{
    bool ok = FALSE;
    int w, h;

    if( filename )
    {
        QApplication :: setOverrideCursor( waitCursor );
        ok = image.load( filename , 0 );
        pm.convertFromImage( image , conversion_flags );

        if( ok )
        {
            setCaption( filename );
            w = pm.width();
            h = pm.height();
            h += menubar->heightForWidth(w) + status->height();
        }
        else
        {
            pm.resize( 0 , 0 );
            update();
        }

        setFixedWidth( w );
        setFixedHeight( h );
        status->setGeometry( 0, height() - status->height(),
                           width(), status->height() );
        QApplication :: restoreOverrideCursor();
    }
    return ok;
}

```

```

}

void ImageViewer :: paintEvent( QPaintEvent *e )
{
    if( pm.size() != QSize( 0, 0 ) )
    {
        QPainter painter( this );
        painter.setClipRect( e->rect() );
        painter.drawPixmap( 0, menubar->heightForWidth(
            width() ), pm );
    }
}

```

Fișierul main.cpp conține instanțierea unui obiect QApplication și a unuia de tip ImageViewer.

```

#include "aplicatie.h"
#include <qapplication.h>
#include <qimage.h>

#ifdef QIMGIO
#include <qimageio.h>
#endif

int main( int argc, char **argv )
{
    QApplication :: setFont( QFont( "Helvetica" , 12 ) );
    QApplication a( argc, argv );

#ifdef QIMGIO
    qInitImageIO();
#endif

    ImageViewer *w = new ImageViewer( 0, "main window",
        QWidget :: WDestructiveClose ) ;
    w->show() ;
    QObject :: connect( qApp, SIGNAL( lastWindowClosed() ),
        qApp, SLOT(quit()) );
    return a.exec() ;
}

```

Lucrarea 3

Îmbunătățirea imaginilor prin operații punctuale

BREVIAR TEORETIC

Termenul general de îmbunătățire a imaginilor se referă la o clasă largă de operații, ce au ca scop mărirea detectabilității componentelor imaginii. Această detectabilitate depinde în primul rând de percepția vizuală a unui observator uman și deci reprezintă o apreciere subiectivă a imaginii.

Îmbunătățirea calității unei imagini se face fără a presupune vreun model de degradare sau a lua în considerare vreo informație legată de imaginea originală. Paradoxal, dar și o imagine originală (nedegradată) poate fi îmbunătățită.

În general, îmbunătățirea se referă la accentuarea unor caracteristici ale imaginii, cum ar fi muchiile, contururile sau contrastul. Procesul de îmbunătățire nu mărește cantitatea de informație conținută de o imagine.

3.1 Operațiile punctuale

Operațiile punctuale sunt definite de o funcție, care atribuie un nou nivel de gri pixelilor din imagine. Noua valoare a pixelului va depinde doar de vechea valoare a acestuia, de unde și denumirea de “operație punctuală”. Matematic se poate scrie:

$$g(k, l) = \phi(f(k, l)) \quad (3.1)$$

unde f este imaginea originală, iar g imaginea îmbunătățită. $g(k, l)$ reprezintă noua valoare a pixelului (k, l) , iar $f(k, l)$ vechea valoare. Operația punctuală este descrisă de funcția ϕ .

O operație sau o transformare punctuală este reprezentată în Figura 3.1.

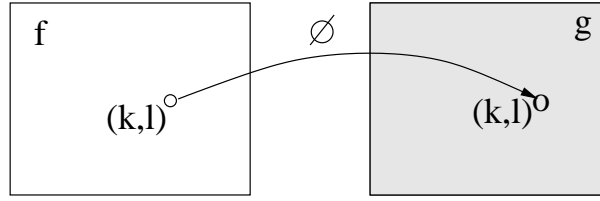


Figura 3.1: Reprezentarea unei operații punctuale.

3.2 Accentuarea contrastului

Această operație se folosește în special pentru îmbunătățirea imaginilor cu contrast scăzut. Acesta poate apărea datorită unei slabe iluminări, a unei iluminări neuniforme sau din cauza unor neliniarități ale caracteristicilor unui senzor de captură a imaginii. O funcție tipică de accentuare a contrastului are următoarea formă matematică:

$$\phi(x) = \begin{cases} \alpha x, & \text{pentru } x \in [0, a) \\ \beta(x - a) + v_a, & \text{pentru } x \in [a, b) \\ \gamma(x - b) + v_b, & \text{pentru } x \in [b, L] \end{cases} \quad (3.2)$$

unde L reprezintă numărul de nivele pe care se face cuantizarea tonurilor de gri. Cazul cel mai frecvent este $L = 256$, pentru o cuantizare pe 8 biți, x luând valori în intervalul $[0, 255]$. Valorile α, β și γ reprezintă pantele celor trei segmente de dreaptă.

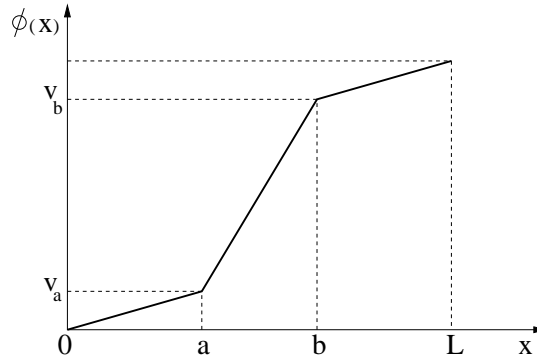


Figura 3.2: Funcția de accentuare de contrast.

După cum se observă din Figura 3.2, pantele α și γ sunt pozitive și subunitare, iar panta β este pozitivă și supraunitară. Un segment cu pantă subunitară realizează o “apropiere” a nivelelor de gri, pe când un segment

cu pantă supraunitară va realiza o “depărtare” a nivelelor de gri, și deci o accentuare a contrastului.

3.3 Întinderea maximă a contrastului

Aceasta este un caz particular al operației de accentuare. Se folosește, în general, pentru reducerea zgomotului dintr-o imagine, atunci când se știe că acesta ia valori cu precădere în intervalul $[a, b]$. Forma matematică este următoarea:

$$\phi(x) = \begin{cases} 0, & \text{pentru } x \in [0, a) \\ \beta(x - a), & \text{pentru } x \in [a, b] \\ L - 1, & \text{pentru } x \in (b, L) \end{cases} \quad (3.3)$$

Nivelele de gri aflate în intervalul $[a, b]$ vor fi distanțate, ca urmare a pantei supraunitare, iar nivelele de gri ce se găsesc în afara acestui interval, vor fi înlocuite fie cu alb, fie cu negru, după caz.

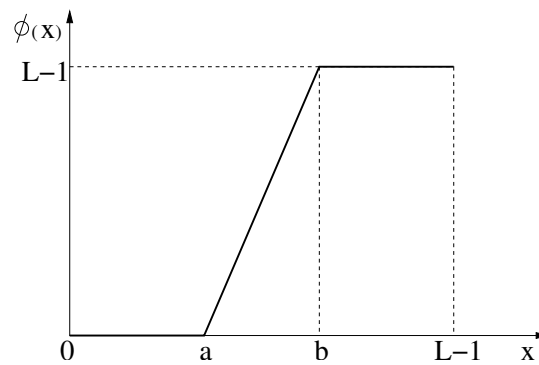


Figura 3.3: Funcția de întindere maximă a contrastului.



Figura 3.4: Întinderea maximă a contrastului: (a) imaginea originală și (b) imaginea rezultată, pentru $a=50$, $b=200$.

3.4 Binarizarea imaginilor

Binarizarea sau prăguirea (“thresholding”) imaginilor este un caz special al întinderii maxime a contrastului, pentru care $a = b$. Rezultatul operației de binarizare este o imagine care conține doar două nivele de gri: alb și negru.

Pentru imagini în tonuri de gri, operația de binarizare se scrie matematic astfel:

$$\phi(x) = \begin{cases} 0, & \text{pentru } x < T \\ L - 1, & \text{pentru } x \geq T \end{cases} \quad (3.4)$$

unde T este o valoare de prag, reprezentând o valoare întreagă din intervalul $[0, L)$.

Pentru imaginile color, binarizarea se poate face în următorul mod:

$$\phi(v) = \begin{cases} 0, & \text{pentru } Y(v) < T \\ L - 1, & \text{pentru } Y(v) \geq T \end{cases} \quad (3.5)$$

unde v este un vector tridimensional ce reprezintă culoarea pixelului (de exemplu $v = (R, G, B)$), iar $Y(v)$ reprezintă luminanța ($Y = 0.3R + 0.6G + 0.1B$).

În urma acestei transformări, contrastul este maximizat la nivelul întregii imagini.

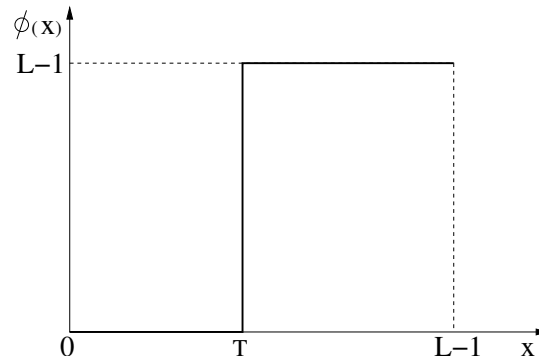


Figura 3.5: Funcția de binarizare.

Figura 3.6: Binarizarea: (a) imaginea originală și (b) imaginea binarizată cu $T=127$.

3.5 Negativarea imaginilor

Negativul unei imagini se obține prin inversarea ordinii nivelelor de gri. Pentru imaginile în tonuri de gri, operația de negativare se face folosind funcția:

$$\phi(x) = (L - 1) - x \quad (3.6)$$

reprezentată în Figura 3.7, iar pentru imaginile color:

$$\phi(v) = ((L - 1) - R, (L - 1) - G, (L - 1) - B) \quad (3.7)$$

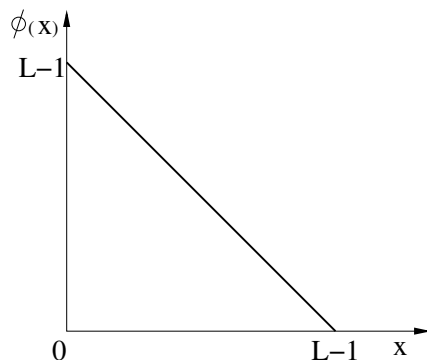


Figura 3.7: Funcția de negativare.

Negativarea imaginilor este utilă pentru afișarea unor imagini medicale sau pentru realizarea de imagini negative pe suporturi fizice, de exemplu, de tip peliculă.



Figura 3.8: Negativarea: (a) imaginea originală și (b) negativul imaginii.

3.6 Decuparea imaginilor

Decuparea cu păstrarea fundalului este dată de formula:

$$\phi(x) = \begin{cases} L-1, & \text{pentru } x \in [a, b] \\ x, & \text{în rest.} \end{cases} \quad (3.8)$$

iar decuparea fără păstrarea fundalului este dată de formula:

$$\phi(x) = \begin{cases} L-1, & \text{pentru } x \in [a, b] \\ 0, & \text{în rest.} \end{cases} \quad (3.9)$$

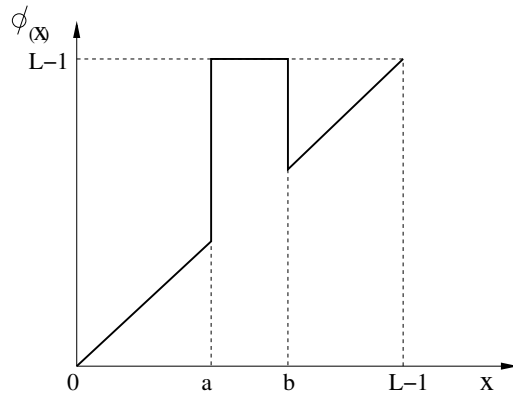


Figura 3.9: Funcția de decupare cu păstrarea fundalului.

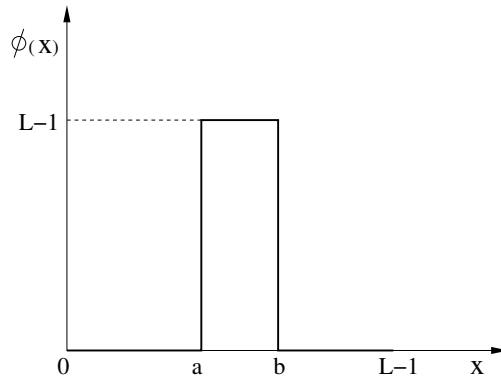


Figura 3.10: Funcția de decupare fără păstrarea fundalului.

Aceste operații permit “decuparea” unor regiuni din imagine, caracterizate de anumite nivele de gri. Acest lucru este folosit atunci când diferite caracteristici ale imaginii sînt conținute în nivelele de gri respective, cum ar fi de exemplu decuparea regiunilor de temperatură joasă reprezentate de nori din imaginile obținute de un satelit meteo. În astfel de imagini, nivelele de gri ce corespund unor nori sînt direct proporționale cu valorile de temperaturi joase.

DESFĂȘURAREA LUCRĂRII

Problema 1. Compilați sursele C++ ale lucrării. Rulați aplicația și

observați rezultatul accentuării contrastului pentru o imagine în tonuri de gri (lena_AN.bmp). Funcția care realizează accentuarea contrastului este prezentată în continuare, pentru următoarele valori: $a = 80$, $b = 170$, $V_a = 20$ și $V_b = 235$.

Pantele care caracterizează funcția de accentuare a contrastului vor fi următoarele: $\alpha = \frac{20}{80} = 0.4$, $\beta = \frac{235-20}{170-80} = \frac{215}{90}$ și $\gamma = \frac{255-235}{255-170} = \frac{20}{85}$.

```
int ImageViewer :: f_accentuare( int nivel_gri )
{
    if( nivel_gri >= 0 && nivel_gri <= 80 )
        return ( int )( 0.4 * nivel_gri );

    if( nivel_gri > 80 && nivel_gri <= 170 )
        return ( int )( 215 / 90. * ( nivel_gri - 80 ) + 20 );

    if( nivel_gri > 170 && nivel_gri <= 255 )
        return ( int )( 20 / 85. * ( nivel_gri - 170 ) + 235 );

    return nivel_gri;
}

void ImageViewer :: accentueaza_contrastul( void )
{
    int w, h;
    int i, j;

    w = image.width();
    h = image.height();

    QImage imag_acc( w, h, 32, 0, QImage :: IgnoreEndian );

    for( i = 0; i < w; i++ )
        for( j = 0; j < h; j++ )
        {
            QRgb pixel = image.pixel( i, j );
            int gri_vechi = qRed( pixel );
            int gri_nou = f_accentuare( gri_vechi );
            imag_acc.setPixel( i, j,
                              qRgb( gri_nou, gri_nou, gri_nou ) );
        }

    QImageIO iio;
    iio.setImage( imag_acc );
}
```

```

iio.setFileName( "imag_acc.bmp" );
iio.setFormat( "BMP" );
iio.write();
}

```

Problema 2. Modificați valorile a, b, V_a și V_b ale funcției de accentuare a contrastului, și observați rezultatele accentuării contrastului.

Problema 3. Implementați operația de întindere maximă a contrastului.

Problema 4. Implementați operația de binarizare. Observați rezultatele acesteia pentru diferite valori ale pragului T .

Problema 5. Observați rezultatul operației de negativare, pentru o imagine în tonuri de gri și pentru o imagine color.

```

void ImageViewer :: negativeaza_imaginea( void )
{
    int w, h;
    int i, j;

    w = image.width();
    h = image.height();

    QImage imag_neg( w, h, 32, 0, QImage :: IgnoreEndian );

    for( i = 0; i < w; i++ )
        for( j = 0; j < h; j++ )
        {
            QRgb pixel = image.pixel( i, j );
            int r = qRed( pixel );
            int g = qGreen( pixel );
            int b = qBlue( pixel );
            imag_neg.setPixel( i, j, qRgb(255-r, 255-g, 255-b) );
        }

    QImageIO iio;
    iio.setImage( imag_neg );
    iio.setFileName( "imag_neg.bmp" );
    iio.setFormat( "BMP" );
    iio.write();
}

```

Problema 6. Implementați cele două operații de decupare.

Lucrarea 4

Îmbunătățirea imaginilor prin egalizarea histogramei

BREVIAR TEORETIC

Tehnicile de îmbunătățire a imaginilor bazate pe calculul histogramei modifică histograma astfel încât aceasta să aibă o anumită formă dorită.

4.1 Histograma unei imagini

Histograma unei imagini reprezintă frecvența relativă de apariție a nivelelor de gri din imagine. Pentru o imagine f , de dimensiune $M \times N$ pixeli, histograma se definește astfel:

$$h(i) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \delta(i, f(m, n)) \quad , \quad i = 0, \dots, L-1 \quad (4.1)$$

unde funcția δ este definită în următorul mod:

$$\delta(x, y) = \begin{cases} 1, & \text{dacă } x = y, \\ 0, & \text{dacă } x \neq y. \end{cases} \quad (4.2)$$

Din punct de vedere statistic, putem considera valoarea fiecărui pixel al imaginii ca o realizare particulară a unei variabile aleatoare asociată nivelelor de gri, caz în care histograma reprezintă funcția de densitate de probabilitate a acestei variabile aleatoare. Fiind o funcție de densitate de probabilitate, histograma oricărei imagini verifică condiția de normare:

$$\sum_{i=0}^{L-1} h(i) = 1 \quad (4.3)$$

Practic, calculul histogramei presupune parcurgerea pixel cu pixel a imaginii și contorizarea numărului de nivele de gri întâlnite.

4.2 Egalizarea histogramei

Egalizarea histogramei reprezintă o operație de accentuare a contrastului și are ca scop obținerea unei histograme uniforme.

Vom asocia unui pixel din imagine o variabilă aleatoare ξ . Astfel, valorile intensității luminoase ale pixelilor reprezintă realizări particulare ale variabilei aleatoare asociate. Vom considera că variabila aleatoare ξ are o densitate de probabilitate $w_\xi(x)$ și o funcție de repartiție $F_\xi(x) = P\{\xi \leq x\}$.

Vom defini în continuare variabila η , care are funcția de repartiție:

$$F_\eta(x) = \int_0^x p_\xi(t) dt$$

și care va fi uniform distribuită în intervalul $(0, 1)$.

Pentru cazul discret, presupunem că nivelele x de gri ale pixelilor au valori între 0 și $L-1$ (unde L este de regulă 256), având asociate probabilitățile de apariție $p_\xi(x_i)$, unde $x_i = 0, 1, \dots, L-1$. Aceste probabilități pot fi estimate pe baza calculului histogramei, considerând imaginea dată, ca fiind o realizare particulară a procesului aleator descris de variabila aleatoare ξ , astfel:

$$p_\xi(x_i) = \frac{h(x_i)}{\sum_{i=0}^{L-1} h(x_i)}$$

Noile nivele de gri, reprezentând valori discrete ale variabilei η din intervalul $[0, L-1]$ se vor calcula cu formulele:

$$h_c(x) = \sum_{x_i=0}^x p_\xi(x_i)$$

$$nivel_{nou} = \text{int} \left[\frac{h_c[nivel_{vechi}] - h_{cmin}}{1 - h_{cmin}} (L-1) + 0.5 \right]$$

unde h_c reprezintă *histograma cumulativă* a imaginii, iar h_{cmin} este valoarea minimă a histogramei cumulative.

4.2.1 Algoritmul de egalizare a histogramei

Algoritmul de egalizare de histogramă, folosit în practică, poate fi descris în limbaj pseudocod astfel:

Pasul 1. Se calculează histograma imaginii:

```

pentru i = 1,...,L
    h[i] = 0
pentru i = 1,...,M
    pentru j = 1,...,N

```

$$\begin{aligned} \text{nivel} &= \text{imagine}[i,j] \\ h[\text{nivel}] &= h[\text{nivel}] + 1 \end{aligned}$$

unde L este numărul de nivele de gri (256), h este histograma imaginii, iar M și N sunt dimensiunile imaginii.

Pasul 2. Se calculează histograma cumulativă a imaginii:

$$\begin{aligned} hc[1] &= h[1] \\ \text{pentru } i &= 2, \dots, L \\ hc[i] &= hc[i-1] + h[i] \end{aligned}$$

Pasul 3. Se calculează noile nivele de gri din imagine, sub forma unei transformări $y = T(x)$ dată de formula:

$$y = T(x) = \left\lceil \frac{hc[x] - hc[1]}{NM - hc[1]}(L - 1) + 0.5 \right\rceil$$

astfel:

$$\begin{aligned} \text{pentru } i &= 1, \dots, M \\ \text{pentru } j &= 1, \dots, N \\ \text{nivel_vechi} &= \text{imagine}[i,j] \\ \text{nivel_nou} &= T(\text{nivel_vechi}) \\ \text{imagine}[i,j] &= \text{nivel_nou} \end{aligned}$$

4.3 Observații

- Deși la prima vedere egalizarea de histogramă ar părea că este o operație punctuală, din cauza formulei de calcul a noilor valori de gri, ea este totuși o operație integrală, datorită faptului că pentru fiecare pixel din imagine noua valoare se calculează pe baza calculului histogramei și, deci, pe baza valorilor tuturor pixelilor din imagine.
- În Figura 4.1 puteți observa cum imaginea a fost îmbunătățită prin egalizarea histogramei.
- În Figura 4.2 puteți observa cum s-a modificat forma histogramei imaginii originale, după egalizare.



Figura 4.1: Egalizarea histogramei: (a) imaginea originală și (b) imaginea rezultată.

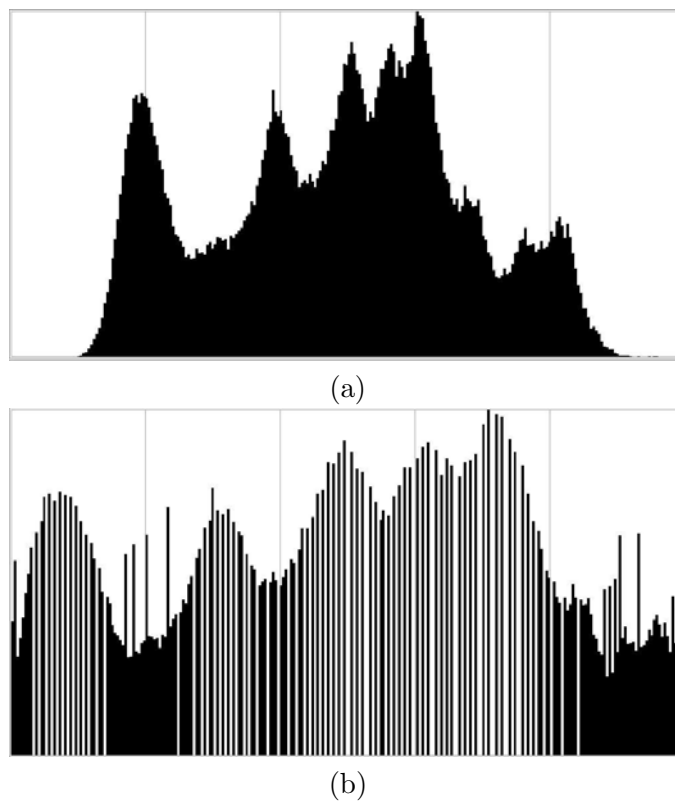


Figura 4.2: Histograma (a) originală și (b) după egalizare.

DESFĂȘURAREA LUCRĂRII

O posibilă implementare a algoritmului de egalizare a histogramei este prezentat în continuare:

```
void ImageViewer :: egalizeaza_histograma( void )
{
    int i, j;
    int width, height;
    int h[ 256 ];

    for( i = 0; i < 256; i++ )
        h[ i ] = 0;

    width = image.width();
    height = image.height();

    //calcularea histogramei imaginii
    for( i = 0; i < width; i++ )
        for( j = 0; j < height; j++ )
        {
            QRgb pixel;
            pixel = image.pixel( i, j );

            int nivel_gri = qRed( pixel );
            h[ nivel_gri ]++ ;
        }

    //calcularea histogramei cumulative
    double hc[ 256 ];

    hc[ 0 ] = h[ 0 ];
    for( i = 1; i < 256; i++ )
        hc[ i ] = hc[ i - 1 ] + h[ i ];

    QImage imag_eq( width, height, 32, 0, QImage::IgnoreEndian );

    //egalizarea histogramei
    for( i = 0; i < width; i++ )
        for( j = 0; j < height; j++ )
        {
            QRgb pixel = image.pixel( i, j );

            int nivel = qRed( pixel );
```

```
int nivel_nou = (int)( ( hc[nivel] - hc[0] ) * 255 /
    ( width*height - hc[0] ) );

imag_eq.setPixel( i, j,
    qRgb( nivel_nou, nivel_nou, nivel_nou ) );
}

image = imag_eq;
pm = image;
update();
}
```

Problema 1. Observați forma histogramei pentru câteva imagini în tonuri de gri.

Problema 2. Observați efectele egalizării de histogramă pentru diferite imagini, inclusiv pentru o imagine subexpusă și pentru una supraexpusă.

Problema 3. Modificați funcția `histograma_imagini` astfel încât aceasta să calculeze histograma cumulativă a imaginii. Observați forma unei histograme cumulative.

Problema 4. Justificați faptul că histograma cumulativă a unei imagini poate fi considerată estimatul unei funcții de repartiție.

Lucrarea 5

Transformări geometrice de bază

BREVIAR TEORETIC

Transformările geometrice sunt transformări care nu modifică valorile pixelilor din imagine, ci modifică doar așezarea lor spațială. Cu alte cuvinte, lasă nemodificată compoziția imaginii, alterându-i însă structura. În urma aplicării unei transformări geometrice asupra unei imagini, histograma acesteia nu se modifică.

5.1 Translația

Translația se definește ca fiind operația de modificare *în linie dreaptă* a coordonatelor unui pixel din imagine de la o poziție la alta. Un pixel de coordonate carteziane (x, y) va avea după translație coordonatele (x', y') , astfel:

$$\begin{cases} x' &= x + T_x \\ y' &= y + T_y \end{cases} \quad (5.1)$$

unde perechea (T_x, T_y) reprezintă vectorul de translație (vezi Figura 5.1).

Translatarea unei imagini se realizează prin translatarea fiecărui pixel în parte. Valorile T_x și T_y sunt numere întregi pozitive sau negative. În cazul în care noile coordonate ale unui pixel depășesc dimensiunile imaginii, atunci el va fi poziționat în partea opusă a imaginii, ca în Figura 5.2.

5.2 Rotația

Rotația se definește ca fiind operația de modificare *după o traiectorie circulară* a coordonatelor unui pixel din imagine (vezi Figura 5.3). Rotația

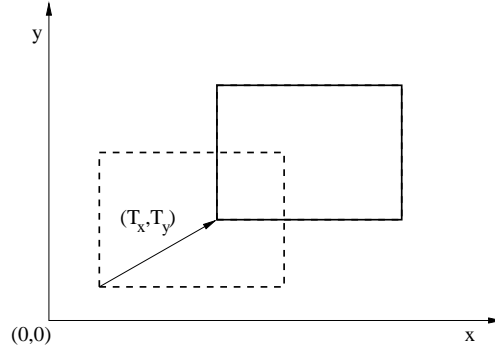


Figura 5.1: Translatarea unui obiect dreptunghiular.

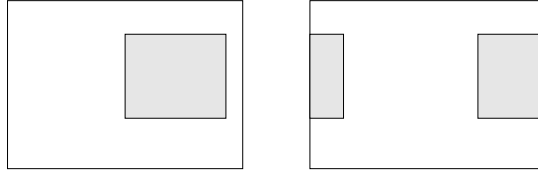


Figura 5.2: Translatarea spre dreapta a unui obiect dreptunghiular într-o imagine.

este specificată de unghiul ϕ . Poziția unui pixel, exprimată în coordonate carteziane (x, y) , se exprimă în coordonate polare (r, θ) astfel:

$$\begin{cases} x &= r \cos \theta \\ y &= r \sin \theta \end{cases} \quad (5.2)$$

Noile coordonate carteziane (x', y') ale pixelului rotit cu un unghi ϕ vor fi:

$$\begin{cases} x' &= r \cos(\theta + \phi) = r \cos \theta \cos \phi - r \sin \theta \sin \phi = x \cos \phi - y \sin \phi \\ y' &= r \sin(\theta + \phi) = r \sin \theta \cos \phi + r \cos \theta \sin \phi = x \sin \phi + y \cos \phi \end{cases} \quad (5.3)$$

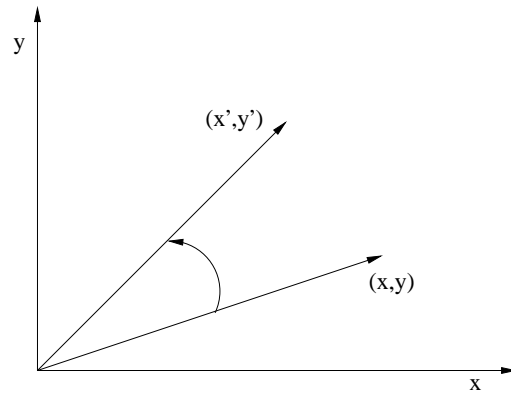


Figura 5.3: Rotația.

5.3 Oglindirea

Oglindirea este operația prin care se produce imaginea în oglindă a unui obiect, relativ la o axă de oglindire. În Figura 5.4 sunt ilustrate cele două feluri de oglindire: oglindirea “stânga-dreapta” și cea “sus-jos”.

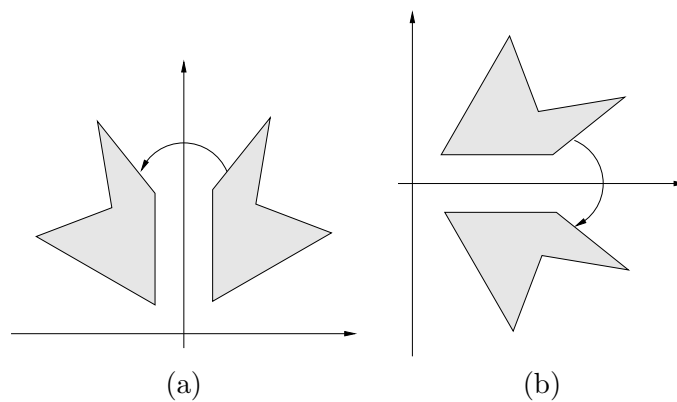


Figura 5.4: Oglindirea (a) “stânga-dreapta” (b) “sus-jos”.

Observați în Figura 5.5 efectele oglinirii “stânga-dreapta” a imaginii Lena, față de o axă verticală ce trece prin centrul imaginii.



Figura 5.5: Ogindirea “stânga-dreapta”: (a) imaginea originală; (b) imaginea oglindită.

DESFĂȘURAREA LUCRĂRII

În continuare este prezentat codul C care implementează translația unei imagini pe orizontală, cu un deplasament de 100 de pixeli. Citiți și înțelegeți codul.

```
void ImageViewer :: translateaza_imaginea( void )
{
    int w, h;
    int i, j;
    int iprim, tx = 100;

    w = image.width();
    h = image.height();

    QImage imag_tx( w, h, 32, 0, QImage :: IgnoreEndian );

    for( i = 0; i < w; i++ )
        for( j = 0; j < h; j++ )
        {
            QRgb pixel = image.pixel( i, j );

            iprim = i + tx;
            if( iprim >= w )
                iprim -= w;
```

```
        imag_tx.setPixel( iprim, j, pixel );
    }

    image = imag_tx;
    pm = image;
    update();
}
```

Problema 1. Implementați operația de translație, pentru următorii doi vectori de translație: $(0,100)$ și $(100,100)$.

Problema 2. Implementați operația de oglindire “stânga-dreapta” față de axa verticală ce trece prin centrul imaginii.

Problema 3. Implementați operația de oglindire “sus-jos” față de axa orizontală ce trece prin centrul imaginii.

Lucrarea 6

Zgomotul în imagini

BREVIAR TEORETIC

Zgomotul este un semnal aleator, care afectează informația utilă conținută într-o imagine. El poate apare de-a lungul unui lanț de transmisiune, sau prin codarea și decodarea imaginii, și reprezintă un element perturbator nedorit. De obicei se încearcă eliminarea lui prin diverse metode de filtrare.

Zgomotul se poate suprapune informației utile în două moduri:

- **aditiv.** În acest caz, zgomotul se numește *zgomot aditiv* și matematic se poate scrie:

$$g(x, y) = f(x, y) + n(x, y) \quad (6.1)$$

unde $f(x, y)$ este imaginea inițială, neafectată de zgomot, $n(x, y)$ este zgomotul, iar $g(x, y)$ este imaginea afectată de zgomot.

- **multiplicativ.** În acest caz zgomotul se numește *zgomot multiplicativ*, iar fenomenul de suprapunere al acestuia peste informația utilă se scrie matematic astfel:

$$g(x, y) = f(x, y) \cdot n(x, y) \quad (6.2)$$

unde $f(x, y)$, $n(x, y)$ și $g(x, y)$ au aceleași semnificații ca mai sus.

În continuare vom trata zgomotul ca fiind aditiv. Modelul de degradare a imaginii este reprezentat în figura 6.1. Zgomotul multiplicativ poate fi tratat la fel de simplu ca zgomotul aditiv, dacă logaritmăm relația (6.2):

$$\log(g(x, y)) = \log(f(x, y) \cdot n(x, y)) = \log(f(x, y)) + \log(n(x, y)) \quad (6.3)$$

Cantitativ, se poate aprecia măsura în care zgomotul a afectat informația utilă, calculând raportul semnal-zgomot¹:

¹SNR = (engl.) Signal Noise Ratio.

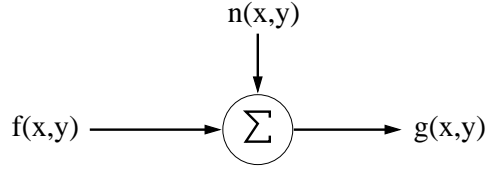


Figura 6.1: Modelul aditiv de degradare.

$$SNR = 10 \log \frac{\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} f^2(i, j)}{\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} n^2(i, j)} [dB] \quad (6.4)$$

$$SNR = 10 \log \frac{\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} f^2(i, j)}{\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} [f(i, j) - g(i, j)]^2} [dB] \quad (6.5)$$

Raportul semnal-zgomot reprezintă raportul dintre energia imaginii originale și energia zgomotului suprapus acesteia. În continuare vom asocia valorilor pe care le ia zgomotul o variabilă aleatoare ξ , care, în funcție de tipul zgomotului, va fi caracterizată de diferite funcții de densitate de probabilitate.

6.1 Zgomotul cu distribuție uniformă

Zgomotul cu distribuție uniformă (vezi Figura 6.2) este caracterizat de o funcție de densitate de probabilitate de forma:

$$w_{\xi}(x) = \begin{cases} A, & \text{pentru } x \in \left[-\frac{k}{2}; \frac{k}{2}\right], \\ 0, & \text{în rest.} \end{cases} \quad (6.6)$$

În Figura 6.3 puteți observa efectele zgomotului cu distribuție uniformă asupra imaginii “Lena”, pentru un raport semnal/zgomot de 5 dB.

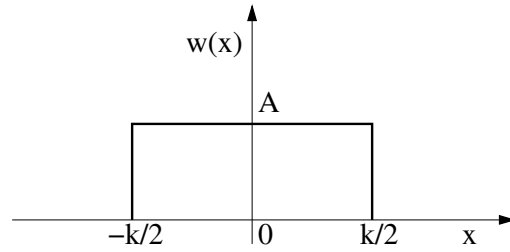


Figura 6.2: Funcția de densitate de probabilitate pentru o distribuție uniformă.



Figura 6.3: Zgomotul uniform: (a) imaginea originală; (b) imaginea afectată de zgomot uniform, SNR=5 dB.

6.2 Zgomotul cu distribuție gaussiană

Zgomotul gaussian este caracterizat de o funcție de densitate de probabilitate de forma (vezi Figura 6.4):

$$w_{\xi}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (6.7)$$

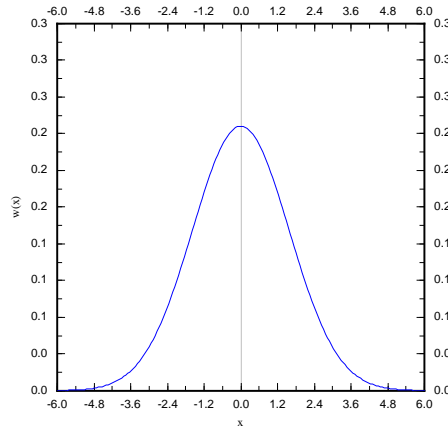


Figura 6.4: Funcția de densitate de probabilitate pentru o distribuție gaussiană.

Valoarea medie a unei variabilei aleatoare ξ , cu o distribuție dată de funcția $w_\xi(x)$ ca în relația (6.7), este μ , iar varianța ei este σ^2 . O distribuție gaussiană se mai numește și *normală*. O distribuție normală de medie μ și varianță σ^2 se notează cu $N(\mu, \sigma^2)$.

În Figura 6.5 puteți observa efectele zgomotului cu distribuție gaussiană asupra imaginii “Lena”, pentru un raport semnal/zgomot de 5 dB.



Figura 6.5: Zgomotul Gaussian: (a) imaginea originală; (b) imaginea afectată de zgomot gaussian, SNR=5 dB.

6.3 Zgomotul de tip “sare și piper”

După cum îi spune numele, acest tip de zgomot va afecta valorile pixelilor în două moduri: “sare” - adică noua valoare a pixelului va fi 255 (pixelul va fi alb), sau “piper” - adică noua valoare a pixelului va fi 0 (pixelul va fi negru). Zgomotul de tip “sare și piper” (vezi Figura 6.6) este perfect decorelat, deoarece între valorile 0 și 255, și între coordonatele pixelilor afectați de zgomot nu există corelație.



Figura 6.6: Zgomotul “sare și piper”: (a) imaginea originală; (b) imaginea cu 10% pixeli afectați de zgomot.

6.4 Alte tipuri de zgomot

Zgomotele diferă între ele în funcție de distribuția care le caracterizează. Alte funcții de distribuție utilizate sunt:

- distribuția Rayleigh: $w_{\xi}(x) = xe^{-\frac{x^2}{2}}$
- distribuția Maxwell: $w_{\xi}(x) = x^2e^{-\frac{x^2}{2}}$
- distribuția Beta: $w_{\xi}(x) = x^b(1-x)^c$
- distribuția Gamma (Erlang): $w_{\xi}(x) = x^ne^{-x}$
- distribuția Laplace: $w_{\xi}(x) = e^{-|x|}$
- distribuția Cauchy: $w_{\xi}(x) = \frac{1}{1+x^2}$

Aceste funcții sunt reprezentate în Figura 6.7.

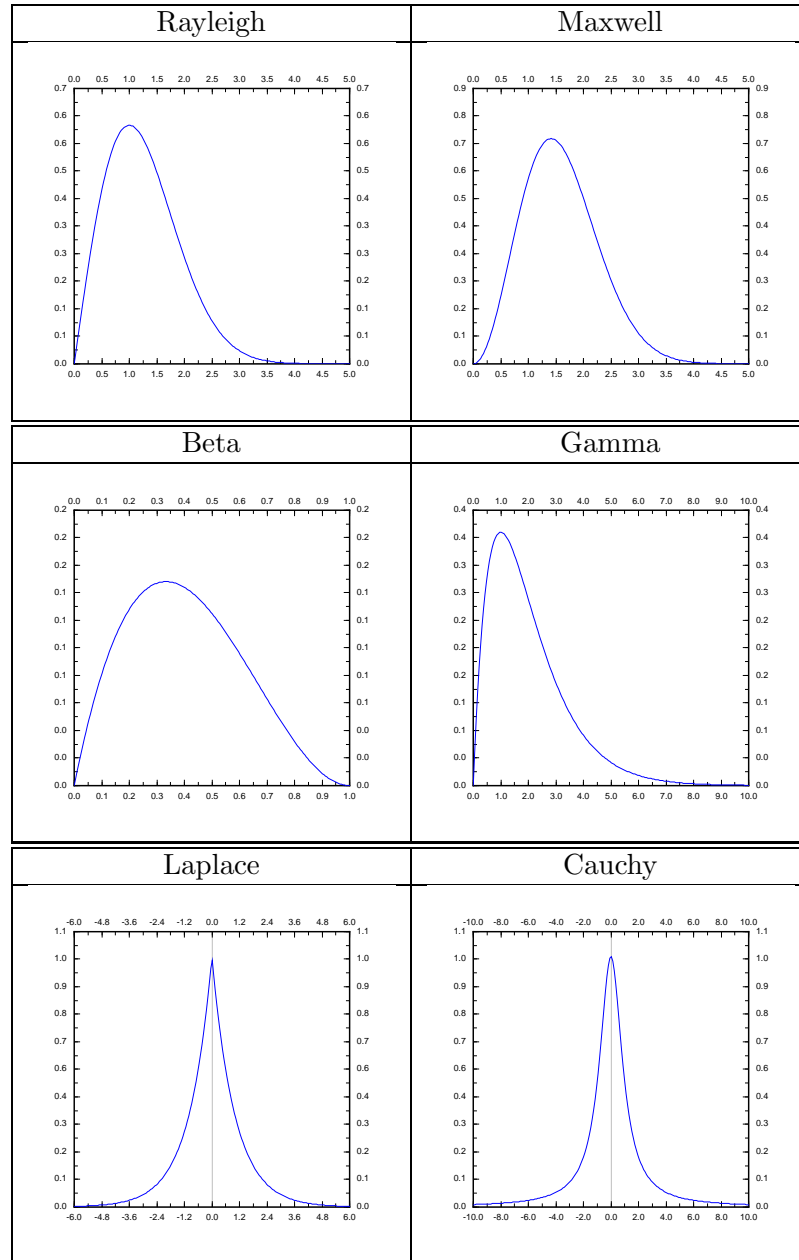


Figura 6.7: Diverse funcții de distribuție.

DESFĂȘURAREA LUCRĂRII

Problema 1. Suprapuneți zgomot cu distribuție uniformă peste o imagine în tonuri de gri (în meniul **Algoritmi**, funcția `zgomot_uniform`).

```

void ImageViewer :: zgomot_uniform( void )
{
    int i, j;

    int w = image.width();
    int h = image.height();

    long int e_zgomot = 0; //energia zgomotului
    long int e_imagine = 0; //energia imaginii

    double SNR; //raportul semnal-zgomot

    //imaginea afectata de zgomot
    QImage image_aff( w, h, 32, 0, QImage::IgnoreEndian );

    int med = 0; //media zgomotului
    int dis = 200; //dispersia zgomotului

    for( i = 0; i < w; i++ )
        for( j = 0; j < h; j++ )
        {
            QRgb pixel = image.pixel( i, j );
            int nivel_gri = qRed( pixel );

            e_imagine += nivel_gri * nivel_gri;

            //srand( rand() );
            int zgomot = ( int )( med + sqrt( 3 ) *
                ( 2. * ( rand() / ( RAND_MAX + 1. )) ) * sqrt(dis) );
            e_zgomot += zgomot * zgomot;
            int val = nivel_gri + zgomot;

            if( val > 255 )
                val = 255;
            if( val < 0 )
                val = 0;

            image_aff.setPixel( i, j, qRgb( val, val, val ) );
        }

    SNR = 10 * log( 1. * e_imagine / e_zgomot );

    image = image_aff;
    pm = image;
}

```

```

update();

QString mesaj;
mesaj.sprintf( "SNR = %6.3lf dB", SNR );
QMessageBox::about( this, "SNR", mesaj );
}

```

Problema 2. Modificați media și dispersia zgomotului cu distribuție uniformă (în fișierul `algoritmi.cpp`, funcția `generează_zgomot_uniform`). Observați efectele.

Problema 3. Suprapuneți zgomot cu distribuție gaussiană peste o imagine în tonuri de gri (în meniul `Algoritmi`, funcția `zgomot_gaussian`). Diferența între această funcție și cea care generează constă în modul în care este calculată valoarea zgomotului, și anume:

```

int zgomot = ( int )( med + sqrt( 2 ) *
    sqrt( -1.* log( rand()/( RAND_MAX + 1. ))) *
    cos( 2 * 3.1415926 * ( rand()/( RAND_MAX+1 ))) *
    sqrt( dis ) );

```

Problema 4. Modificați media și dispersia zgomotului cu distribuție gaussiană (fișierul `algoritmi.cpp`, funcția `generează_zgomot_gaussian`). Observați efectele.

Problema 5. Suprapuneți zgomot de tip “sare și piper” peste o imagine în tonuri de gri (în meniul `Algoritmi`, funcția `zgomot_salt_and_pepper`). Codul acesteia este prezentat în continuare:

```

void ImageViewer :: salt_and_pepper( void )
{
    int i, j, k;
    int w = image.width();
    int h = image.height();

    double nr = 0.1; //procentul de pixeli afectati de zgomot

    srand( rand() );

    k = 0;
    while( k < ( int )( w * h * nr ) )
    {
        i = ( int )( 1. * w * rand() / ( RAND_MAX + 1. ) );
        j = ( int )( 1. * h * rand() / ( RAND_MAX + 1. ) );

        QRgb sare_piper;
    }
}

```



```
    if( ( 100. * rand() / ( RAND_MAX + 1. ) ) >= 50 )
        sare_piper = qRgb( 255, 255, 255 );
    else
        sare_piper = qRgb( 0, 0, 0 );

    if( (i >= 0) && (i < w) && (j >= 0) && (j < h) )
        image.setPixel( i, j, sare_piper );

    k++;
}

pm = image;
update();
}
```

Problema 6. Modificați procentul de pixeli afectați de zgomot de tip “sare și piper” (fișierul `algoritmi.cpp`, funcția `zgomot_salt_and_pepper`). Observați efectele.

Lucrarea 7

Filtrarea imaginilor

BREVIAR TEORETIC

Filtrarea imaginilor se înscrie în clasa operațiilor de îmbunătățire, principalul scop al acesteia fiind eliminarea zgomotului suprapus unei imagini.

Filtrarea reprezintă o operație de vecinătate. Prin aceasta se înțelege că la calculul noii valori a unui pixel vor contribui și valorile pixelilor vecini, nu doar vechea lui valoare, cum se întâmpla la operațiile punctuale. Vecinii unui pixel reprezintă o mulțime de pixeli, aflați în apropierea acestuia, care alcătuiesc o *vecinătate*. Această vecinătate poate avea diverse forme și dimensiuni, însă cele mai utilizate în practică sînt vecinătățile de formă pătrată, de dimensiuni impare.

7.1 Filtrarea liniară a imaginilor

După cum îi spune numele, acest tip de filtrare respectă principiul liniarității (sau al superpoziției).

Principiul liniarității: Fiind date două imagini $f_1(x, y)$ și $f_2(x, y)$, și două numere reale α și β , se numește **operator liniar**, un operator O care are următoarea proprietate:

$$O[\alpha \cdot f_1(x, y) + \beta \cdot f_2(x, y)] = \alpha \cdot O[f_1(x, y)] + \beta \cdot O[f_2(x, y)] \quad (7.1)$$

Operația de filtrare liniară calculează noua valoare a unui pixel al imaginii, din poziția (m, n) , ca o combinație liniară a unui număr de valori din imaginea originală, astfel:

$$g(m, n) = \sum_{(k, l) \in W} w_{kl} \cdot f(m - k, n - l) \quad (7.2)$$

unde $f(x, y)$ este imaginea originală (afectată sau nu, de zgomot), $g(x, y)$ este imaginea filtrată, W este o structură de puncte care definește vecinătatea

pixelului (m, n) , w_{kl} sînt niște valori constante care reprezintă **coeficienții filtrului**.

Filtrul este definit de vecinătatea W și de coeficienții w_{kl} . Un filtru poate fi specificat de o matrice V , care poartă numele de **mască de convoluție** sau **mască de filtrare**, care este caracterizată de formă, valorile coeficienților și de origine. În Figura 7.1 este prezentată o mască de filtrare de formă pătrată, de dimensiune 3×3 , având originea în centru.

$w_{-1,-1}$	$w_{-1,0}$	$w_{-1,1}$
$w_{0,-1}$	$w_{0,0}$	$w_{0,1}$
$w_{1,-1}$	$w_{1,0}$	$w_{1,1}$

Figura 7.1: Mască de filtrare pătrată de dimensiune 3×3 .

Nu este obligatoriu ca forma măștii de filtrare să fie pătrată, de dimensiune impară sau să aibă originea în centrul măștii.

Operația de filtrare liniară poate fi descrisă astfel: se suprapune masca de filtrare peste fiecare pixel al imaginii originale, astfel încât originea măștii să coincidă cu pixelul considerat, apoi se calculează toate produsele între coeficienții măștii și valorile pixelilor peste care se suprapun acești coeficienți, iar suma acestor produse reprezintă noua valoare a pixelului considerat. Această tehnică poartă numele de **tehnica ferestrei glisante**. Operația descrisă reprezintă de fapt o convoluție bidimensională.

7.1.1 Filtrele de netezire

Filtrele de netezire sunt echivalentele bidimensionale ale filtrelor trece-jos (FTJ), și la fel ca acestea, sînt folosite în general pentru eliminarea zgomotului, care se presupune că este de bandă largă.

Informația conținută într-o imagine, în general, se regăsește în componentele de joasă frecvență, și deci este justificată o filtrare trece-jos pentru reducerea puterii zgomotului.

Zgomotul din imagine se presupune că este aditiv și pur aleator, adică se consideră următoarele ipoteze simplificatoare:

- $g(i, j) = f(i, j) + n(i, j)$ (zgomotul n este aditiv),
- n este un semnal staționar (comportamentul său statistic nu depinde de coordonatele i și j ale pixelului),

- $\bar{n} = 0$ (media zgomotului este zero),
- dacă zgomotul are dispersia σ_n , atunci:

$$\overline{n(i_1, j_1) \cdot n(i_2, j_2)} = \begin{cases} \sigma_n^2 & \text{pentru } i_1 = i_2 \text{ \& } j_1 = j_2 \\ 0 & \text{în rest} \end{cases}$$

(zgomotul este complet decorelat).

Condiția de normare a coeficienților filtrelor de netezire

Pentru un filtru trece-jos se impune următoarea condiție: componenta continuă a imaginii să nu fie alterată de filtru. Cu alte cuvinte, filtrul să conserve luminozitatea medie a imaginii.

Pentru aceasta, considerăm o imagine având un singur nivel de gri, constant, notat cu μ , adică: $f(i, j) = \mu$ pentru oricare i și j . Pentru ca filtrul să conserve luminozitatea medie, adică valoarea medie μ , impunem $g(i, j) = \mu$ pentru $\forall i, j$. Rezultă:

$$\mu = \sum_{(k,l) \in W} w_{kl} \cdot \mu \quad (7.3)$$

$$\sum_{(k,l) \in W} w_{kl} = 1 \quad (7.4)$$

unde $w_{kl} \geq 0$.

Relația (7.4) poartă numele de condiție de normare pentru filtre de netezire (sau trece-jos).

Filtrul de mediere

Filtrul de mediere este cel mai simplu filtru de netezire. Caracteristic unui filtru de mediere este faptul că toți coeficienții măștii de filtrare sînt egali. Dacă ținem cont și de condiția de normare, atunci coeficienții unui filtru de mediere, care are o mască de filtrare de dimensiune $N \times N$, au valoarea $\frac{1}{N^2}$. În Figurile 7.2, 7.3 și 7.4 sunt prezentate măștile de filtrare de mediere, pentru $N = 3, 5$ și respectiv 7 .

$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$

Figura 7.2: Mască de filtrare pătrată de dimensiune 3×3 .

$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$
$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$
$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$
$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$
$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$

Figura 7.3: Mască de filtrare pătrată de dimensiune 5×5 .

$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$
$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$
$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$
$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$
$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$
$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$
$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$

Figura 7.4: Mască de filtrare pătrată de dimensiune 7×7 .

(a)



(b)

Figura 7.5: Filtrarea de mediere: (a) imaginea originală; (b) imaginea filtrată cu o mască 7×7 .

Filtrul de mediere nu este folosit în practică, deoarece, pe lângă zgomot, are de suferit și semnalul util (vezi Figura 7.5). Filtrarea de mediere este deranjantă pentru imagine în acele zone în care imaginea conține frecvențe înalte (variații bruște), pentru că duce la apariția fenomenului de încetșoare¹.

¹(engl.) blurring.

Din punctul de vedere al zgomotului, pentru filtrare este utilă o mască de filtrare de dimensiune cât mai mare. Din punctul de vedere al semnalului util, al imaginii, este util ca masca să fie cât mai mică. În practică se realizează un compromis între cele două aspecte.

Alte măști de filtrare

Alte măști de filtrare, pentru filtrele de mediere, sunt prezentate în continuare, deși toate au același randament nesatisfăcător.

$$\begin{pmatrix} \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\ \frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\ \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \end{pmatrix} \begin{pmatrix} \frac{1}{16} & \frac{1}{16} & \frac{1}{16} \\ \frac{1}{16} & \frac{1}{2} & \frac{1}{16} \\ \frac{1}{16} & \frac{1}{16} & \frac{1}{16} \end{pmatrix} \begin{pmatrix} 0 & \frac{1}{5} & 0 \\ \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\ 0 & \frac{1}{5} & 0 \end{pmatrix} \begin{pmatrix} 0 & \frac{1}{8} & 0 \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ 0 & \frac{1}{8} & 0 \end{pmatrix}$$

7.1.2 Filtrele trece-sus

Filtrele trece-sus urmăresc eliminarea componentelor de frecvență joasă din imagine. Sunt folosite în general pentru detectarea frontierelor sau contururilor din imagine, acolo unde au loc treceri sau variații bruște ale luminanței.

Condiția de normare a coeficienților filtrelor trece-sus

Condiția de normare a coeficienților pentru un filtru trece-jos (relația (7.6)) se determină impunând condiția ca filtrul să rejeteze complet (sau să atenueze complet) componenta continuă a imaginii.

Pentru aceasta vom considera, la fel, o imagine $f(i, j) = \mu$ pentru $\forall i, j$. La ieșirea filtrului trece-sus vom avea $g(i, j) = 0$ pentru $\forall i, j$.

$$0 = \sum_{(k,l) \in W} w_{kl} \cdot \mu \quad (7.5)$$

$$\sum_{(k,l) \in W} w_{kl} = 0 \quad (7.6)$$

Filtrul de accentuare

Filtrul de accentuare nu este un filtru trece-sus, ci folosește filtrarea trece-sus pentru a realiza accentuarea. Prin accentuare se înțelege contrastarea unei imagini și are ca scop îmbunătățirea percepției vizuale a contururilor obiectelor. Cu alte cuvinte, îmbunătățirea detectabilității componentelor scenei de-a lungul frontierelor acestora. Acest lucru se realizează, în principiu, prin modificarea valorilor pixelilor situați de o parte și de alta a unei frontiere comune.

Sistemul uman are tendința de a adânci profilul zonelor de tranziție dintre regiuni uniforme. Studiul fiziologiei sistemului vizual a demonstrat că aceasta se realizează prin prelucrări de tip derivativ ce apar în diferitele etape pe care le parcurge informația vizuală. Efectul global poate fi modelat matematic prin scăderea din semnalul original a unei derivate secunde ponderate.

În continuare sunt prezentate câteva măști de implementare a unei derivate secunde de tip Laplace:

$$\begin{pmatrix} 0 & -\frac{1}{4} & 0 \\ -\frac{1}{4} & 1 & -\frac{1}{4} \\ 0 & -\frac{1}{4} & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \end{pmatrix} \begin{pmatrix} -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} \\ -\frac{1}{8} & 1 & -\frac{1}{8} \\ -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} \end{pmatrix}$$

În Figura 7.6 puteți observa efectele filtrării Laplace.

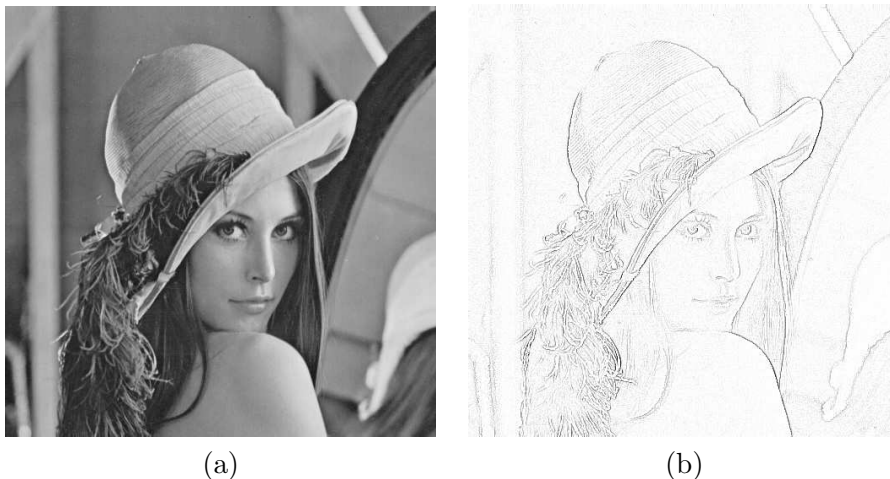


Figura 7.6: Filtrarea Laplace: (a) imaginea originală; (b) imaginea rezultată (negativată).

Filtrul de accentuare se implementează după schema prezentată în Figura 7.7.

7.2 Filtrarea neliniară a imaginilor

Filtrele neliniare nu respectă principiul liniarității sau al superpoziției enunțat la începutul lucrării. Acestea au apărut din necesitatea de a depăși limitările filtrelor liniare, în ceea ce privește zgomotele care nu au o distribuție normală sau nu sunt aditive.

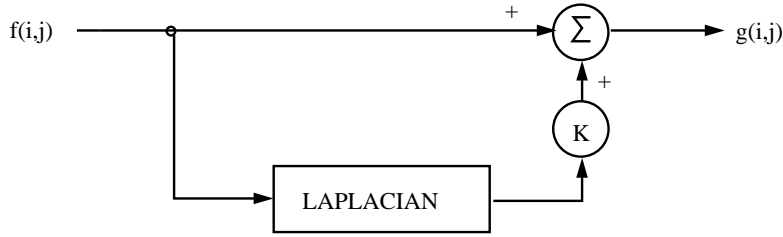


Figura 7.7: Filtrul de accentuare.

7.2.1 Filtrele de ordine

Filtrele de ordine sunt operatori locali, definiți la rândul lor de o fereastră, care selectează din imagine un număr de pixeli vecini pixelului curent, într-un mod identic cu tehnica ferestrei glisante. Valorile pixelilor selectați se ordonează crescător.

Să presupunem că fereastra conține n pixeli, iar valorile lor formează următoarea mulțime:

$$\{x_1, x_2, \dots, x_n\} \quad (7.7)$$

După ce aceste valori au fost ordonate crescător, vom avea:

$$\{x_{(1)}, x_{(2)}, \dots, x_{(n)}\} \quad (7.8)$$

pentru care sunt îndeplinite următoarele condiții:

$$x_{(1)} \leq x_{(2)} \leq \dots \leq x_{(n)} \quad (7.9)$$

Ieșirea filtrului de ordine de ordin k , pentru $k \in [1; n]$ întreg, este statistica de ordinul k , cu alte cuvinte, elementul de pe poziția k din șirul ordonat:

$$\text{rank}_k\{x_1, x_2, \dots, x_n\} = x_{(k)} \quad (7.10)$$

Filtrul median

Filtrul median este un filtru de ordine a cărui ieșire este statistica de ordin central, adică elementul ce se află pe poziția de mijloc a șirului ordonat de valori selectate de fereastra de filtrare:

$$\text{median}\{x_1, x_2, \dots, x_n\} = \begin{cases} x_{(\frac{n+1}{2})} & \text{dacă } n \text{ este impar,} \\ \frac{1}{2}x_{(\frac{n}{2})} + \frac{1}{2}x_{(\frac{n}{2}+1)} & \text{dacă } n \text{ este par.} \end{cases} \quad (7.11)$$

Filtrul median este potrivit pentru eliminarea zgomotului de tip “sare și piper”. După ordonarea valorilor pixelilor, valorile zgomotului (adică 0 sau 255) se vor situa pe primele, respectiv ultimele poziții în mulțime, și deci, la ieșirea filtrului, vom avea o valoare diferită de valorile zgomotului. Totuși există și situații în care, după filtrare, mai există pixeli afectați de zgomot. În acest caz, spunem că filtrul a fost “străpuns” de zgomot. Acest lucru este posibil atunci când mai mult de jumătate din pixelii selectați de fereastra de filtrare, sunt afectați în același mod (sare sau piper, 255 sau 0) de zgomot.



(a)



(b)

Figura 7.8: Filtrul median: (a) imaginea originală afectată de zgomot “sare și piper”; (b) imaginea filtrată.

Filtrul de minim

Filtrul de minim este un filtru de ordine a cărui ieșire este statistica de ordinul 1, adică valoarea $x_{(1)}$, care este cea mai mică valoare din mulțimea de valori ale pixelilor selectați de către fereastra de filtrare.

Filtrul de maxim

Filtrul de maxim este un filtru de ordine a cărui ieșire este statistica de ordinul n , adică valoarea $x_{(n)}$, care este cea mai mare valoare din mulțimea de valori luate în considerare.

DESFĂȘURAREA LUCRĂRII

Problema 1. Observați efectele filtrului de mediere pentru o imagine afectată de zgomot gaussian. Codul C al funcției care implementează filtrul de

mediere este prezentat în continuare:

```
void ImageViewer :: filtru_mediere( void )
{
    int i, j;
    int k, l;
    int w, h;

    double v[ 3 ][ 3 ];

    //coeficientii mastii de filtrare
    v[0][0] = 1./9; v[0][1] = 1./9; v[0][2] = 1./9;
    v[1][0] = 1./9; v[1][1] = 1./9; v[1][2] = 1./9;
    v[2][0] = 1./9; v[2][1] = 1./9; v[2][2] = 1./9;

    w = image.width();
    h = image.height();

    QImage image_fil( w, h, 32, 0, QImage::IgnoreEndian );

    for( i = 1; i < w - 1; i++ )
        for( j = 1; j < h - 1; j++ )
        {
            //suma ponderata
            double sum = 0;

            for( k = -1; k < 2; k++ )
                for( l = -1; l < 2; l++ )
                    sum += v[ k + 1 ][ l + 1 ] *
                        qRed( image.pixel( i + k, j + l ) );

            image_fil.setPixel( i, j,
                                qRgb( (int)sum, (int)sum, (int)sum ) );
        }

    image = image_fil;
    pm = image;
    update();
}
```

Problema 2. Observați efectele filtrului de mediere pentru o imagine afectată de zgomot de tip “sare și piper”.

Problema 3. Observați efectul de “blurring” al filtrului de mediere pentru o imagine neafectată de zgomot.

Problema 4. Implementați un filtru de mediere cu o mască de filtrare de formă pătrată de dimensiune 5x5.

Problema 5. Observați efectul filtrului de accentuare (pentru o imagine neafectată de zgomot). Codul C al filtrului este următorul:

```
void ImageViewer :: filtru_accentuare( void )
{
    int i, j;
    int k, l;
    int w, h;

    double sum;
    double v[ 3 ][ 3 ];

    //coeficientii mastii
    v[0][0] = 0;      v[0][1] = -1./4; v[0][2] = 0;
    v[1][0] = -1./4; v[1][1] = 1;      v[1][2] = -1./4;
    v[2][0] = 0;      v[2][1] = -1./4; v[2][2] = 0;

    w = image.width();
    h = image.height();

    QImage image_fil( w, h, 32, 0, QImage::IgnoreEndian );

    for( i = 1; i < w - 1; i++ )
        for( j = 1; j < h - 1; j++ )
        {
            sum = 0;

            for( k = -1; k < 2; k++ )
                for( l = -1; l < 2; l++ )
                    sum += 1. * v[ k + 1 ][ l + 1 ] *
                        qRed( image.pixel( i + k, j + l ) );

            int niv = qRed( image.pixel( i, j ) );
            niv = (int)( niv + 0.6 * sum );
            image_fil.setPixel( i, j, qRgb( niv, niv, niv ) );
        }

    image = image_fil;
    pm = image;
    update();
}
```

Problema 6. Observați efectele filtrului median pentru o imagine afectată de zgomot de tip “sare și piper”. Citiți și înțelegeți implementarea în C:

```
void ImageViewer :: filtru_median( void )
{
    int i, j;
    int w, h;
    int k, aux;
    int m, n;
    int med;
    int sir[ 9 ];

    w = image.width();
    h = image.height();

    QImage image_fil( w, h, 32, 0, QImage::IgnoreEndian );

    for( i = 1; i < w-1; i++ )
        for( j = 1; j < h-1; j++ )
        {
            //formarea unui sir din elementele vecinatatii 3x3
            k = 0;
            for( m = -1; m < 2; m++ )
                for( n = -1; n < 2; n++ )
                {
                    sir[k] = qRed( image.pixel( i+m, j+n ) );
                    k++;
                }

            //ordonarea crescatoare a valorilor pixelilor
            //metoda BUBBLE SORT
            k = 0;
            while( k == 0 )
            {
                k = 1;
                for( m = 0; m < 8; m++ )
                    if( sir[ m ] > sir[ m + 1 ] )
                    {
                        aux = sir[ m ];
                        sir[ m ] = sir[ m + 1 ];
                        sir[ m + 1 ] = aux;
                        k = 0;
                    }
            }
        }
    }
```

```
    }

    //elementul median
    med = sir[ 4 ];

    //noua valoare a pixelului
    image_fil.setPixel( i, j, qRgb( med, med, med ) );
}

image = image_fil;
pm = image;
update();
}
```

Problema 7. Implementați filtrul de minim. Observați efectele lui asupra unei imagini neafectate de zgomot.

Problema 8. Implementați filtrul de maxim. Observați efectele lui asupra unei imagini neafectate de zgomot.

Lucrarea 8

Transformări unitare

BREVIAR TEORETIC

Transformările reprezintă o categorie de prelucrări ce include operații de tip integral, la calculul noii valori a unui pixel al imaginii transformate contribuind valorile tuturor pixelilor din imaginea originală.

Termenul de transformare se referă la o clasă de matrici unitare folosite pentru a reprezenta imagini. O matrice A de dimensiune $N \times N$ este unitară dacă inversa ei este matricea A^{*T} :

$$A \cdot A^{-1} = A \cdot A^{*T} = A^{*T} \cdot A = I_N \quad (8.1)$$

unde $*$ reprezintă operația de complementare în mulțimea numerelor complexe, T reprezintă operația de transpunere a unei matrici, iar I_N este matricea identitate de dimensiune $N \times N$:

$$I_N = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix} \quad (8.2)$$

Pentru un vector uni-dimensional \underline{u} de dimensiune N , de forma:

$$\underline{u} = \begin{bmatrix} u(0) \\ u(1) \\ \vdots \\ \vdots \\ u(N-1) \end{bmatrix} = [u(0), u(1), \dots, u(N-1)]^T \quad (8.3)$$

se numește **transformare unitară directă** relația:

$$v(k) = \sum_{n=0}^{N-1} a(k, n) \cdot u(n), \quad 0 \leq k \leq N-1 \quad (8.4)$$

unde $v(k)$ reprezintă elementele vectorului transformat \underline{v} , iar $a(k, n)$ sînt elementele matricii A . Matricial această relație se poate scrie astfel:

$$\underline{v} = A \cdot \underline{u} \quad (8.5)$$

Transformarea unitară inversă este dată de relația:

$$u(n) = \sum_{k=0}^{N-1} v(k) \cdot a^*(k, n), \quad 0 \leq n \leq N-1 \quad (8.6)$$

care se scrie matricial astfel:

$$\underline{u} = A^{*T} \cdot \underline{v} \quad (8.7)$$

Valorile vectorului \underline{v} sunt o reprezentare a vectorului inițial \underline{u} , într-un alt spațiu. O astfel de reprezentare este utilă în filtrare, compresie, extragere de trăsături sau alte tipuri de analiză a imaginilor.

8.1 Transformări unitare bidimensionale

Pentru o imagine $u(m, n)$, de dimensiune $N \times N$, transformarea directă are următoarea formă:

$$v(k, l) = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} u(m, n) \cdot a_{k,l}(m, n), \quad 0 \leq k, l \leq N-1 \quad (8.8)$$

iar transformarea inversă:

$$u(m, n) = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} v(k, l) \cdot a_{k,l}^*(m, n), \quad 0 \leq m, n \leq N-1 \quad (8.9)$$

unde coeficienții $\{a_{k,l}(m, n)\}$ poartă numele de **transformare unitară bidimensională**, și reprezintă un set de matrici de bază ortonormale, iar $v(k, l)$ reprezintă transformata imaginii $u(m, n)$.

Aceste matrici de bază respectă condiția de **ortonormalitate**:

$$\sum_{m=0}^{N-1} \sum_{n=0}^{N-1} a_{k,l}(m, n) \cdot a_{k',l'}^*(m, n) = \delta(k - k', l - l') = \begin{cases} 1, & k = k' \text{ \& } l = l', \\ 0, & \text{în rest.} \end{cases} \quad (8.10)$$

pentru $\forall k, l$.

O transformare ca cea dată de relația (8.8) este caracterizată de N^4 coeficienți $a_{k,l}(m, n)$. Pentru calculul unui singur element $v(k, l)$ (k și l fixați) este nevoie de un număr de N^2 înmulțiri. Prin urmare complexitatea întregului calcul este $O(N^4)$. Complexitatea poate fi redusă la $O(N^3)$ dacă transformarea este separabilă.

O transformare este **separabilă**, dacă elementele $a_{k,l}(m, n)$ ce o definesc, pot fi scrise ca produs de alte două elemente, grupate după perechi de indici, astfel:

$$a_{k,l}(m, n) = a_k(m) \cdot b_l(n) = a(k, m) \cdot b(l, n) \quad (8.11)$$

unde, matricile $A = \{a(k, m)\}$ și $B = \{b(l, n)\}$ trebuie să fie la rândul lor unitare, adică:

$$A \cdot A^{*T} = A^{*T} \cdot A = I_N \quad (8.12)$$

$$B \cdot B^{*T} = B^{*T} \cdot B = I_N \quad (8.13)$$

Dacă transformarea este separabilă, atunci relațiile (8.8) și (8.9) devin:

$$v(k, l) = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} a(k, m) \cdot u(m, n) \cdot b(l, n) \quad (8.14)$$

$$u(m, n) = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} a^*(k, m) \cdot v(k, l) \cdot b^*(l, n) \quad (8.15)$$

care pot fi scrise matricial astfel:

$$V = A \cdot U \cdot B^T \quad (8.16)$$

$$U = A^{*T} \cdot V \cdot B^* \quad (8.17)$$

unde $U = \{u(m, n)\}$ reprezintă imaginea originală, iar $V = \{v(k, l)\}$ reprezintă imaginea transformată.

În practică se folosesc numai transformări separabile, pentru care, în plus, se alege $B = A$. În acest caz, vom avea o singură matrice A , unitară, care definește transformarea, iar relațiile (8.14) și (8.15) devin:

$$v(k, l) = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} a(k, m) \cdot u(m, n) \cdot a(l, n) \quad (8.18)$$

$$u(m, n) = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} a^*(k, m) \cdot v(k, l) \cdot a^*(l, n) \quad (8.19)$$

Matricial, relațiile (8.16) și (8.17) se scriu după cum urmează:

$$V = A \cdot U \cdot A^T \quad (8.20)$$

$$U = A^{*T} \cdot V \cdot A^* \quad (8.21)$$

8.1.1 Proprietățile transformărilor unitare

În continuare vor fi prezentate câteva din proprietățile transformărilor unitare.

- O transformare unitară conservă energia semnalului. Această proprietate o vom demonstra pentru cazul unei transformări unitare unidimensionale, pentru simplitate, deși este perfect valabilă și în cazul unei transformări bidimensionale. Fie \underline{u} un semnal discret unidimensional, format din N eșantioane, și o transformare unitară dată de matricea A . Relațiile de transformare vor fi:

$$\begin{aligned} \underline{v} &= A \cdot \underline{u} \\ \underline{u} &= A^{*T} \cdot \underline{v} \end{aligned}$$

Energia semnalului \underline{u} este dată de norma la pătrat a spațiului în care este reprezentat semnalul:

$$E_v = \|\underline{v}\|^2 = \underline{v}^{*T} \cdot \underline{v} = (A\underline{u})^{*T} \cdot A\underline{u} = \underline{u}^{*T} A^{*T} A \underline{u} = \underline{u}^{*T} \cdot \underline{u} = \|\underline{u}\|^2 = E_u \quad (8.22)$$

În general transformarea se alege astfel încât energia să fie inegal distribuită în spațiul transformatei, chiar dacă ea era uniform distribuită în spațiul original.

- Entropia unui semnal discret cu valori aleatoare se conservă printr-o transformare unitară. Dar entropia este o măsură a cantității de informație, ceea ce înseamnă că o transformare unitară păstrează informația conținută în semnal.
- Coeficienții în spațiul transformatei sunt decorelați sau aproape decorelați. Transformata optimă care compactează maximum de energie într-un număr dat de coeficienți și care în același timp decorelează complet acești coeficienți, este transformarea Karhunen-Loève.

8.2 Transformata Fourier discretă

8.2.1 Transformata Fourier unidimensională

Pentru un semnal unidimensional, \underline{u} , de dimensiune N , de forma:

$$\underline{u} = [u(0), u(1), \dots, u(N-1)]^T \quad (8.23)$$

transformarea Fourier directă este dată de relația:

$$v(k) = \sum_{n=0}^{N-1} u(n) \cdot e^{-\frac{2\pi jkn}{N}} \quad \forall k = 0..N-1 \quad (8.24)$$

iar transformarea Fourieri inversă de relația:

$$u(n) = \frac{1}{N} \sum_{k=0}^{N-1} v(k) \cdot e^{\frac{2\pi jkn}{N}} \quad \forall n = 0..N-1 \quad (8.25)$$

Astfel definită, transformarea Fourier nu este unitară. Următoarele relații definesc transformarea Fourier unitară, directă și inversă:

$$v(k) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} u(n) \cdot e^{-\frac{2\pi jkn}{N}} \quad \forall k = 0..N-1 \quad (8.26)$$

$$u(n) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} v(k) \cdot e^{\frac{2\pi jkn}{N}} \quad \forall n = 0..N-1 \quad (8.27)$$

Dacă definim matricea $F = \{f(k, n)\}$ a transformării, având elementele:

$$f(k, n) = \frac{1}{\sqrt{N}} e^{-\frac{2\pi jkn}{N}} \quad k, n = 0..N-1 \quad (8.28)$$

atunci transformarea Fourier se poate scrie matricial astfel:

$$\underline{v} = F \cdot \underline{u} \quad (8.29)$$

$$\underline{u} = F^* \cdot \underline{v} \quad (8.30)$$

cu observația că matricea F are următoarea proprietate: $F = F^T$.

Pentru calculul transformatei Fourier discrete, există algoritmi rapizi (FFT¹) care reduc complexitatea calculului de la $O(N^2)$ la $O(N \log N)$.

8.2.2 Transformarea Fourier bidimensională

Pentru o imagine $U = \{u(m, n)\}_{m,n=0..N-1}$, de dimensiune $N \times N$, imaginea transformată $V = \{v(k, l)\}_{k,l=0..N-1}$ se calculează cu relația următoare, ce reprezintă transformarea Fourier în ipoteza separabilității:

$$v(k, l) = \frac{1}{N} \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} u(m, n) \cdot e^{-\frac{2\pi j(km+ln)}{N}} \quad (8.31)$$

¹Fast Fourier Transform

iar transformarea Fourier inversă este dată de formula:

$$u(m, n) = \frac{1}{N} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} v(k, l) \cdot e^{\frac{2\pi j(km+ln)}{N}} \quad (8.32)$$

Dacă folosim matricea F definită cu relația (8.28), atunci matricial se poate scrie:

$$V = F \cdot U \cdot F \quad (8.33)$$

$$U = F^* \cdot V \cdot F^* \quad (8.34)$$

8.3 Transformata cosinus discretă

Transformata cosinus este o transformată unitară separabilă, definită de matricea $C = \{c(k, n)\}$, ale cărei elemente sunt date de relația:

$$c(k, n) = \begin{cases} \frac{1}{\sqrt{N}}, & k = 0, 0 \leq n \leq N-1 \\ \sqrt{\frac{2}{N}} \cos \frac{(2n+1)\pi k}{2N}, & 1 \leq k \leq N-1, 0 \leq n \leq N-1 \end{cases} \quad (8.35)$$

Transformata cosinus, directă și inversă, pentru un semnal unidimensional, este dată de relațiile:

$$v(k) = \alpha(k) \sum_{n=0}^{N-1} u(n) \cos \frac{(2n+1)\pi k}{2N}, \quad 0 \leq k \leq N-1 \quad (8.36)$$

$$u(n) = \sum_{k=0}^{N-1} \alpha(k) v(k) \cos \frac{(2n+1)\pi k}{2N}, \quad 0 \leq n \leq N-1 \quad (8.37)$$

unde

$$\alpha(0) = \sqrt{\frac{1}{N}}, \quad \alpha(k) = \sqrt{\frac{2}{N}}, \quad 1 \leq k \leq N-1 \quad (8.38)$$

Transformarea cosinus bidimensională, directă și inversă, este dată de următoarele două relații, scrise matricial:

$$V = C \cdot U \cdot C^T \quad (8.39)$$

$$U = C^T \cdot V \cdot C \quad (8.40)$$

deoarece matricea C are proprietatea că $C = C^*$, elementele sale fiind numere reale.

Observație: transformarea cosinus nu este partea reală a transformării Fourier.

8.4 Transformata sinus discretă

Transformata sinus este o transformată unitară separabilă, definită de matricea $S = \{s(k, n)\}$, ale cărei elemente sunt date de relația:

$$s(k, n) = \sqrt{\frac{2}{N+1}} \sin \frac{(k+1)(n+1)\pi}{N+1}, \quad 0 \leq k, n \leq N-1 \quad (8.41)$$

Relațiile ce definesc transformarea sinus unidimensională, directă și inversă, sunt următoarele:

$$v(k) = \sqrt{\frac{2}{N+1}} \sum_{n=0}^{N-1} u(n) \sin \frac{(k+1)(n+1)\pi}{N+1}, \quad 0 \leq k \leq N-1 \quad (8.42)$$

$$u(n) = \sqrt{\frac{2}{N+1}} \sum_{k=0}^{N-1} v(k) \sin \frac{(k+1)(n+1)\pi}{N+1}, \quad 0 \leq n \leq N-1 \quad (8.43)$$

Transformarea sinus bidimensională, directă și inversă, se scrie matricial astfel:

$$V = S \cdot U \cdot S \quad (8.44)$$

$$U = S \cdot V \cdot S \quad (8.45)$$

deoarece matricea S are proprietatea că $S = S^* = S^T = S^{-1}$.

Observație: Transformarea sinus nu este partea imaginară a transformării Fourier.

DESFĂȘURAREA LUCRĂRII

Problema 1. Pentru o imagine de dimensiune $N \times N$, adică pătrată, observați imaginea transformată obținută cu ajutorul transformatei cosinus bidimensională (funcția `transformata_cosinus_discreta` din meniul `Algoritmi`). Codul acestei funcții este prezentat în continuare:

```
void ImageViewer :: transformata_cosinus_discreta( void )
{
    int w, h;
    int i, j, k;
    double pi = 3.1415926;

    w = image.width();
```

```

h = image.height();

if( w == h )
{
    int N = w;
    double max = 0;
    double C[ N ][ N ];
    //matricea transformarii cosinus

    int U[ N ][ N ];
    //matricea imaginii in spatiul original

    double V[ N ][ N ];
    //matricea imaginii in spatiul transformatei

    double  AUX[ N ][ N ];

    //formarea matricei C a transformarii cosinus discreta
    for( i = 0; i < N; i++ )
        C[ 0 ][ i ] = 1. / sqrt( N );

    for( i = 1; i < N; i++ )
        for( j = 0; j < N; j++ )
        {
            C[ i ][ j ] = sqrt( 2./N ) *
                cos( pi * ( 2*j + 1 ) * i / ( 2*N ) );

            if( C[ i ][ j ] > max )
                max = C[ i ][ j ];
        }

    //formarea matricei U
    for( i = 0; i < N; i++ )
        for( j = 0; j < N; j++ )
            U[ i ][ j ] = qRed( image.pixel( i, j ) );

    //V = C*U*Ct
    //mai intii vom calcula AUX = C * U
    for( i = 0; i < N; i++ )
        for( j = 0; j < N; j++ )
        {
            AUX[ i ][ j ] = 0;
            for( k = 0; k < N; k++ )
                AUX[ i ][ j ] += C[ i ][ k ] * U[ k ][ j ];
        }

```

```

    }

    //apoi V = AUX * Ct
    max = 0;
    for( i = 0; i < N; i++ )
        for( j = 0; j < N; j++ )
        {
            V[ i ][ j ] = 0;
            for( k = 0; k < N; k++ )
                V[ i ][ j ] += AUX[ i ][ k ] * C[ j ][ k ];

            if( V[ i ][ j ] > max )
                max = V[ i ][ j ];
        }

    QImage transf( N, N, 32, 0, QImage::IgnoreEndian );

    for( i = 0; i < N; i++ )
        for( j = 0; j < N; j++ )
        {
            int niv = (int)( V[ i ][ j ] * 255 / max );
            transf.setPixel( i, j, qRgb( niv, niv, niv ) );
        }

    image = transf;
    pm = image;
    update();
}

```

Problema 2. Implementați transformarea sinus bidimensională.

Problema 3. Implementați transformarea Fourier bidimensională (numai partea reală, care reprezintă spectrul de frecvențe spațiale al imaginii).

Problema 4. Verificați proprietatea de conservare a energiei pentru una din transformări.

Lucrarea 9

Compresia imaginilor

BREVIAR TEORETIC

Termenul de compresie se referă la totalitatea metodelor ce au drept scop reducerea cantității de date necesare pentru reprezentarea unei imagini. Compresia este folosită în special pentru stocarea sau transmiterea imaginilor.

Să considerăm cazul unei imagini de dimensiune 512×512 pixeli. Dacă aceasta este o imagine în tonuri de gri, iar fiecare pixel este codat cu 8 biți, atunci cantitatea de date necesară pentru a reprezenta această imagine este:

$$512 \times 512 \times 8 = 2^9 \times 2^9 \times 2^3 = 2^{21} \approx 2 \text{ Mb}$$

Din acest calcul ne putem da seama că pentru a stoca o imagine avem nevoie de spațiu considerabil, iar pentru transmiterea ei avem nevoie de un canal de transmisiune de bandă largă, de care nu dispunem întotdeauna.

9.1 Clasificarea metodelor de compresie

Metodele de compresie se pot clasifica astfel:

- **Metode de compresie la nivel de pixel**

Aceste metode nu țin cont de corelația care există între pixelii vecini, codând fiecare pixel ca atare. Acest tip de compresie este fără pierdere de informație, adică imaginea inițială poate fi refăcută perfect din imaginea comprimată. Exemple de astfel de metode:

- codarea Huffman
- codarea LZW (Lempel-Ziv-Walsh)
- codarea RLE (Run Length Encoding)

- **Metode de compresie predictive**

Aceste metode realizează compresia folosind corelația care există între pixelii vecini, dintr-o imagine. Exemple de astfel de metode:

- codarea cu modulație “delta”
- codarea DPCM (Differential Pulse Code Modulation)
- **Metode de compresie cu transformate**
 Aceste metode se bazează pe scrierea imaginii într-o altă bază, prin aplicarea unei transformări unitare, astfel încât energia imaginii să fie concentrată într-un număr cât mai mic de coeficienți.
- **Alte metode de compresie**
 - cuantizarea vectorială
 - codarea folosind fractali
 - codarea hibridă

9.2 Algoritmul Huffman

Să presupunem că valorile pixelilor unei imagini sunt simboluri ale unei surse S :

$$[S] = \{S_1, S_2, \dots, S_N\} \quad (9.1)$$

pentru care se cunosc probabilitățile de apariție:

$$[P] = \{p_1, p_2, \dots, p_N\} \quad (9.2)$$

$$P(S_1) = p_1 \quad P(S_2) = p_2 \quad P(S_N) = p_N \quad (9.3)$$

Aceste probabilități nu reprezintă altceva decât frecvențele relative de apariție ale simbolurilor într-un șir de simboluri emise de sursa S .

Entropia sursei S care generează simbolurile se calculează cu formula:

$$H(S) = - \sum_{i=1}^N p_i \cdot \log p_i \quad (9.4)$$

Ne propunem să codăm simbolurile sursei S cu simboluri ale unei alte surse (de exemplu o sursă care generează doar două simboluri: 0 și 1), astfel încât entropia noii surse să fie maximizată.

În continuare este prezentată o metodă care maximizează această entropie, metodă elaborată de Huffman în 1952:

Pasul 1. Se ordonează descrescător probabilitățile p_i .

Pasul 2. Se formează un arbore binar, având ca frunze valorile celor mai mici probabilități din șirul de probabilități. Rădăcina acestui arbore va conține suma probabilităților celor două frunze ale sale. Se etichetează muchia stângă cu 1 și muchia dreaptă cu 0.

Pasul 3. Din șirul P se elimină cele două probabilități care au fost alese ca fiind cele mai mici. În șirul P se introduce valoarea conținută de rădăcina arborelui binar format.

Pasul 4. Dacă în șirul P există mai mult de un element, atunci se reia algoritmul, de la **Pasul 1**.

Pasul 5. Codarea binară a fiecărui element se obține prin parcurgerea arborelui ce s-a format, de la rădăcină spre fiecare frunză.

Eficiența codificării Huffman este dată de lungimea medie \bar{l} a cuvintelor de cod, care se calculează folosind formula:

$$\bar{l} = \sum_{i=1}^N l_i \cdot p_i \quad (9.5)$$

unde l_i este lungimea codului alocat simbolului S_i .

Exemplu:

Fie o sursă S care generează 4 simboluri, $[S] = \{a, b, c, d\}$, care au următoarele probabilități de apariție: $[P] = \{0.2; 0.4; 0.1; 0.3\}$. Arborele codării Huffman se construiește conform etapelor prezentate în Figurile 9.1, 9.2, 9.3 și 9.4.



Figura 9.1: Algoritmul Huffman: etapa 1.

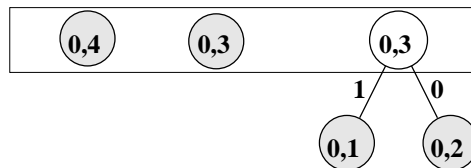


Figura 9.2: Algoritmul Huffman: etapa 2.

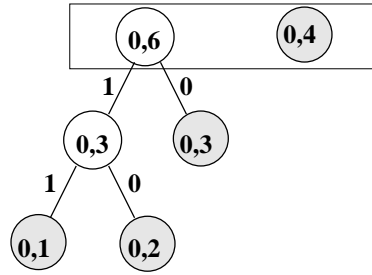


Figura 9.3: Algoritm Huffman: etapa 3.

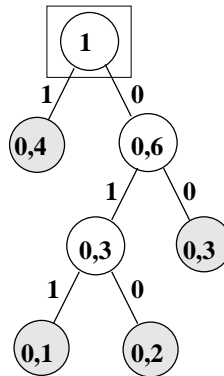


Figura 9.4: Algoritm Huffman: etapa 4.

Pentru decompresie este necesară o tabelă în care să se memoreze corespondențele între simboluri și cuvintele de cod. Fără aceasta decompresia este imposibil de realizat.

Simbol	Cuvânt de cod
a	"010"
b	"1"
c	"011"
d	"00"

Lungimea medie a cuvintelor de cod, pentru acest exemplu, este:

$$\bar{l} = \sum_{i=1}^4 p_i \cdot l_i = 0,2 \cdot 3 + 0,4 \cdot 1 + 0,1 \cdot 3 + 0,3 \cdot 2 = 1,9 \text{ bits/simbol}$$

Dacă nu am fi codat simbolurile, în vederea maximizării entropiei sursei, ar fi fost nevoie de 2 biți/simbol pentru codare.

Pentru imagini, probabilitățile de apariție ale nivelelor de gri se obțin prin calcularea histogramei imaginii. Dacă histograma este uniformă, atunci algoritmul Huffman de codare nu este eficient, nerealizând nici o îmbunătățire a lungimii cuvintelor de cod.

9.3 Algoritmul RLE

9.3.1 Algoritmul RLE pentru imagini binare

Vom considera valorile pixelilor (0 sau 255) ca fiind simbolurile 0 și 1 generate de o sursă binară. În vederea codării imaginea este transformată într-un șir unidimensional, prin concatenarea liniilor sau a coloanelor, ca în Figura 9.5.

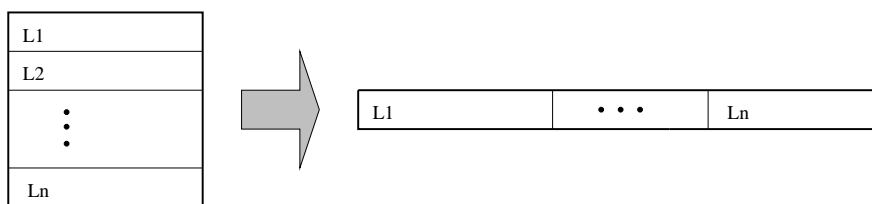


Figura 9.5: Transformarea imaginii într-un șir unidimensional, prin concatenarea liniilor.

Acest șir de elemente 0 și 1 va fi codat, codarea realizându-se astfel: primul element al șirului codat este primul element din șirul de codat; apoi se scrie în șirul codat lungimea fiecărui subșir constant din șirul de codat.

Exemplu:

șirul de codat: 0000000111110001100000000010100011111111111

șirul codat: 0 7 5 3 2 9 1 1 1 3 12

Acest tip de codare se folosește în special pentru comprimarea imaginilor transmise prin fax.

Decompresia se face similar cu compresia, parcurgând șirul codat și generând șiruri alternate, de simboluri 0 sau 1, începând cu primul element din șirul codat, și de lungimi indicate de valorile întâlnite în șirul de decodat.

9.3.2 Algoritmul RLE pentru imagini în tonuri de gri

Pentru imagini în tonuri de gri, algoritmul RLE se aplică pentru plane formate din biții de pe aceeași poziție, din reprezentarea binară a valorilor pixelilor. De exemplu, dacă imaginea în tonuri de gri, are 256 de nivele de gri, corespunzător la o cuantizare pe 8 biți, atunci din această imagine se

construiesc 8 plane (sau 8 imagini binare) astfel: o imagine binară formată din biții b_0 , o altă imagine binară din biții b_1 , ș.a.m.d. (vezi Figura 9.6).

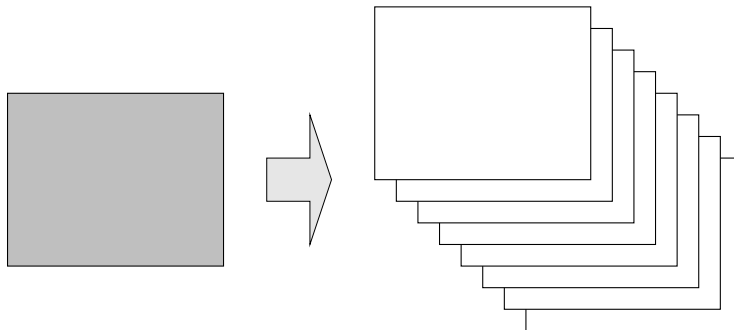


Figura 9.6: Transformarea unei imagini cu 256 nivele de gri, în 8 imagini binare.

Valoarea pixelului (i, j) din imaginea în tonuri de gri va fi reprezentată pe 8 biți astfel:

$$val(i, j) = [b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7]$$

unde b_0 este cel mai semnificativ bit (MSB¹), iar b_7 este cel mai puțin semnificativ bit (LSB²).

Imaginea binară formată din biții cei mai semnificativi va fi comprimată cel mai bine cu algoritmul RLE. Imaginea binară formată din biții cei mai puțin semnificativi va fi o imagine cu “purici”, pentru care se poate lua decizia de a nu mai fi codată și deci ignorată.

9.4 Compresia cu transformate

Compresia cu ajutorul transformatelor se bazează pe proprietatea acestora de a compacta energia imaginii într-un număr redus de coeficienți, cât mai decorelați, repartizați neuniform în spațiul transformării.

Formula care definește o transformarea directă este următoarea:

$$V = A \cdot U \cdot A^T \quad (9.6)$$

unde A este matricea ce definește o transformare unitară, separabilă.

Pentru compresia imaginilor, transformarea cea mai apropiată din punct de vedere al performanțelor de transformare optimă Karhunen-Loève, este transformarea cosinus. Coeficienții de energie mare sunt situați în colțul

¹Most Significant Bit.

²Least Significant Bit.

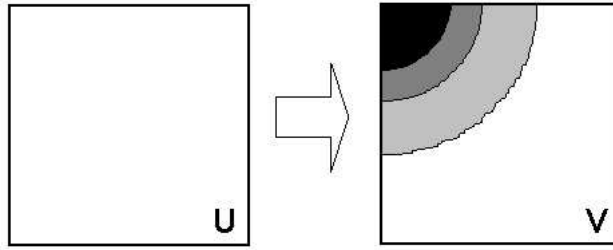


Figura 9.7: Transformarea directă.

din stânga-sus al imaginii transformate, în cazul în care se folosește pentru compresie transformarea cosinus (vezi Figura 9.7).

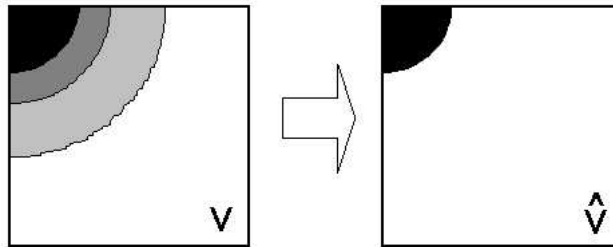


Figura 9.8: Anularea coeficienților de energie mică.

Pentru a obține o rată de compresie mai mare, vor fi anulați coeficienții de energie mică (vezi Figura 9.8). Anularea acestor coeficienți va duce, însă, la scăderea calității imaginii după decompresie. Adică, prin transformare inversă (vezi Figura 9.9), imaginea obținută din imaginea \hat{V} , nu va fi exact imaginea originală U .

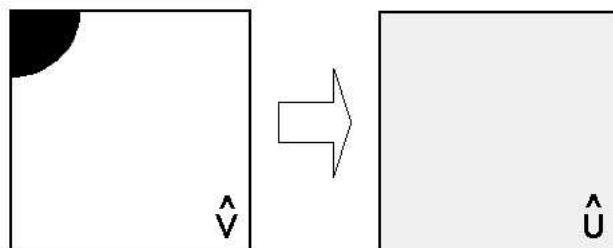


Figura 9.9: Transformarea inversă.

$$\hat{U} = A^{*T} \cdot \hat{V} \cdot A^* \quad (9.7)$$

Compresia cu transformarea cosinus stă la baza algoritmului JPEG³ de compresie a imaginilor.

DESFĂȘURAREA LUCRĂRII

Problema 1. Implementați în C unul dintre algoritmi prezentați (Huffman sau RLE). Pentru simplitate, imaginea citită în format BMP, va fi scrisă într-un format simplificat (vezi funcția `write_naked_image`, din meniul `Algoritmi`). Imaginea comprimată va fi scrisă într-un fișier cu extensia `.huf` sau `.rle`.

```
void ImageViewer :: write_naked_image( void )
{
    int i, j;
    int w, h;
    FILE *file;

    w = image.width();
    h = image.height();

    file = fopen( "naked.img", "w" );
    if( file != NULL )
    {
        for( i = 0; i < w; i++ )
        {
            for( j = 0; j < h; j++ )
            {
                int niv = qRed( image.pixel( i, j ) );
                //nivelul de gri al pixelului

                fprintf( file, "%3d ", niv );
            }
            fprintf( file, "\n" );
        }

        fclose( file );
    }
}
```

Problema 2. Calculați raportul de compresie obținut, ca raport dintre dimensiunile celor două fișiere: cel original (`naked.img`) în formatul simplificat și cel comprimat.

³Joint Photographic Experts Group.

Problema 3. Observați efectul suprimării coeficienților de energie joasă, la o compresie-decompresie folosind transformata cosinus discretă. (funcția `compresie_decompresie_cu_DCT`, din meniul `Algoritmi`). Pentru aceasta vizualizați imaginea `decompresata.bmp`. Codul prezentat în continuare presupune că imaginea este pătrată:

```
void ImageViewer :: compresie_decompresie_cu_DCT( void )
{
    int w, h;
    int i, j, k;
    double pi = 3.1415926;

    w = image.width();
    h = image.height();

    int N = w;
    double max;
    double C[ N ][ N ]; //matricea transformarii cosinus
    double U[ N ][ N ]; //imaginea in spatiul original
    double V[ N ][ N ]; //imaginea in spatiul transformatei
    double AUX[ N ][ N ];

    // COMPRESIA IMAGINII
    //formarea matricei C a transformarii cosinus discreta
    for( i = 0; i < N; i++ )
        C[ 0 ][ i ] = 1. / sqrt( N );

    for( i = 1; i < N; i++ )
        for( j = 0; j < N; j++ )
            C[ i ][ j ] = sqrt( 2./N ) *
                cos( pi * ( 2*j + 1 ) * i / ( 2*N ) );

    //formarea matricei U
    for( i = 0; i < N; i++ )
        for( j = 0; j < N; j++ )
            U[ i ][ j ] = qRed( image.pixel( i, j ) );

    //V = C*U*Ct
    //mai intii vom calcula AUX = C * U
    for( i = 0; i < N; i++ )
        for( j = 0; j < N; j++ )
        {
            AUX[ i ][ j ] = 0;
            for( k = 0; k < N; k++ )
```

```

        AUX[ i ][ j ] += C[ i ][ k ] * U[ k ][ j ];
    }

    //apoi V = AUX * Ct
    max = 0;
    for( i = 0; i < N; i++ )
        for( j = 0; j < N; j++ )
        {
            V[ i ][ j ] = 0;
            for( k = 0; k < N; k++ )
                V[ i ][ j ] += AUX[ i ][ k ] * C[ j ][ k ];

            if( V[ i ][ j ] > max )
                max = V[ i ][ j ];
        }

    //anularea coeficientilor
    //in vederea maririi factorului de compresie
    for( i = 0; i < N; i++ )
        for( j = 0; j < N; j++ )
        {
            if( V[ i ][ j ] < 100 ) //pragul de anulare
                V[ i ][ j ] = 0;

            //alte valori prag: -500, -100, 0, 100, 500
        }

    // DECOMPRESIA IMAGINII
    //U = Ct * V * C
    //AUX = Ct * V
    for( i = 0; i < N; i++ )
        for( j = 0; j < N; j++ )
        {
            AUX[ i ][ j ] = 0;
            for( k = 0; k < N; k++ )
                AUX[ i ][ j ] += C[ k ][ i ] * V[ k ][ j ];
        }

    //apoi U = AUX * C
    for( i = 0; i < N; i++ )
        for( j = 0; j < N; j++ )
        {
            U[ i ][ j ] = 0;
            for( k = 0; k < N; k++ )

```

```

        U[ i ][ j ] += AUX[ i ][ k ] * C[ k ][ j ];
    }

    //pseudo-imaginea diferenta
    QImage diff( N, N, 32, 0, QImage:: IgnoreEndian );

    for( i = 0; i < N; i++ )
        for( j = 0; j < N; j++ )
        {
            int dif = abs( qRed( image.pixel( i, j ) ) -
                (int)( U[ i ][ j ] ) );
            diff.setPixel( i, j, qRgb( dif, dif, dif ) );
        }

    iio.setImage( diff );
    iio.setFileName( "diferenta.bmp" );
    iio.setFormat( "BMP" );
    iio.write();
}

```

Problema 4. Calculați eroarea pătratică medie (ε) dintre imaginea originală și imaginea obținută prin compresia și decompresia cu transformată cosinus discretă, folosind formula:

$$\varepsilon = \overline{(U - V)^2} = \frac{1}{NM} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} [u(i, j) - v(i, j)]^2$$

Pentru aceasta modificați funcția **Compresie-decompresie** cu DCT. Observați valorile erorii medii pătratice pentru diverse valori ale pragului de anulare a coeficienților în spațiul transformării.

Problema 5. Observați pseudo-imaginea diferență, dintre imaginea originală și cea obținută prin compresia și decompresia cu DCT, pentru diferite valori ale pragului (vezi fișierul `diferenta.bmp`).

Lucrarea 10

Segmentarea imaginilor

BREVIAR TEORETIC

Segmentarea reprezintă împărțirea imaginii în zone de interes, după anumite criterii. Fiecărui pixel i se va atribui o valoare, 0 sau 1, reprezentând apartenența acestuia la o anumită zonă sau regiune de interes. De regulă, segmentarea urmărește extragerea, identificarea sau recunoașterea unui anumit obiect dintr-o imagine. Zonele sau regiunile care alcătuiesc o imagine poartă numele de segmente. Pentru o imagine $f(m, n)$, segmentarea reprezintă împărțirea lui f într-un număr N de zone $f_i(m, n)$, cu $i = 1..N$, ca în Figura 10.1. Aceste segmente se numesc complete, dacă au următoarele proprietăți:

- $f_i \cap f_j = \emptyset$ pentru $i \neq j$,
- $\bigcup_{i=1}^N f_i = f$,
- segmentul f_i să fie compact, pentru $\forall i$,
- pentru $\forall i$, un anumit criteriu de uniformitate $E(f_i)$ este satisfăcut,
- pentru $\forall i, j$, criteriul de uniformitate pentru $f_i \cup f_j$ nu este satisfăcut.

Metodele de segmentare a imaginilor se pot clasifica în:

- metode de segmentare orientate pe regiuni
- metode de segmentare orientate pe contururi

10.1 Segmentarea orientată pe regiuni

În general, operația de segmentare orientată pe regiuni urmărește extragerea din imagine a zonelor (regiunilor) ocupate de diversele obiecte prezente

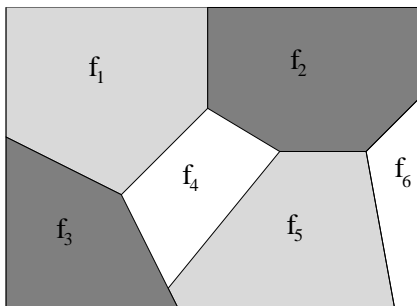


Figura 10.1: Exemplu teoretic de segmentare.

în scenă. Un obiect se definește ca o entitate caracterizată de un set de parametri ale căror valori nu se modifică în diferitele puncte ce aparțin entității considerate. Unul dintre cei mai simpli parametri de definiție este nivelul de gri al pixelului. Dacă nivelul de gri caracterizează în mod suficient obiectele din imagine, atunci histograma imaginii va prezenta o structură de moduri dominante - adică de intervale de nivele de gri ce apar cu probabilitate mai mare. Fiecare mod al histogramei va reprezenta câte un obiect sau o categorie de obiecte.

10.1.1 Prăguirea histogramei

Separarea modurilor histogramei, și deci identificarea obiectelor din imagine, se face prin alegerea unor nivele de gri, numite praguri de segmentare. De obicei aceste praguri se aleg ca fiind corespunzătoare minimelor locale ale histogramei. Din imaginea inițială f se construiește o imagine de etichete g (imagine etichetată), conform unei transformări de forma:

$$g(m, n) = \begin{cases} E_0, & 0 \leq f(m, n) < T_{K-1} \\ E_K, & T_{K-1} \leq f(m, n) < L \end{cases} \quad (10.1)$$

În cazul unei histogramă bimodale (care conține două moduri dominante), ca cea din Figura 10.2(a), transformarea de mai sus devine:

$$g(m, n) = \begin{cases} E_0, & 0 \leq f(m, n) < T \\ E_1, & T \leq f(m, n) < L \end{cases} \quad (10.2)$$

asemănătoare cu operația de binarizare. O astfel de histogramă bimodală este caracteristică imaginilor ce conțin un singur obiect sau mai multe obiecte de același fel, pe un fundal uniform. De exemplu, o imagine care conține un scris negru pe un fond alb.

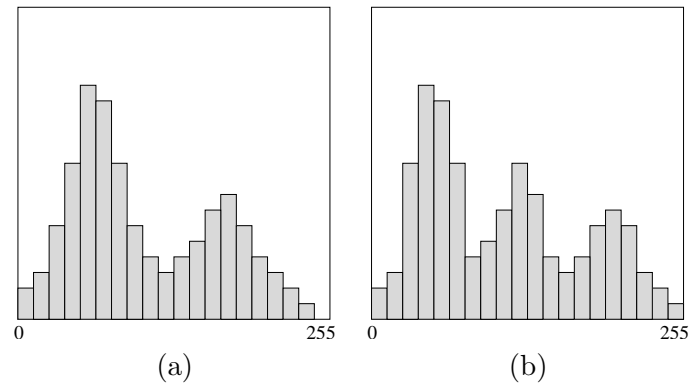


Figura 10.2: Histogramă: (a) bimodală; (b) cu trei moduri dominante.

10.1.2 Segmentarea prin creștere de regiuni

Principiul pe care se bazează creșterea regiunilor este următorul: se aleg în imagine pixeli reprezentativi pentru fiecare obiect individual, pe baza cărora are loc un proces de aglomerare a pixelilor vecini acestora, ce au aceleași proprietăți cu pixelii reprezentativi. În urma acestui proces de aglomerare se obțin zone de pixeli cu aceleași caracteristici.

Procesul se oprește în momentul în care fiecare pixel al imaginii a fost alocat unei regiuni. Metoda are două etape esențiale: alegerea punctelor de start (puncte inițiale), numite *germeni* sau *semințe*, și creșterea propriu-zisă a regiunilor. Numărul final de regiuni rezultate este egal cu numărul de germeni aleși inițial pentru creștere. În principiu, este de dorit ca fiecare obiect individual aflat în imagine, să fie marcat de un germene. Dacă în interiorul unui obiect se găsesc mai mulți germeni, pentru fiecare dintre ei va fi crescută o regiune, fapt ce duce la o segmentare artificială nedorită a obiectului respectiv. Acest neajuns poate fi corectat printr-o etapă de fuziune a regiunilor adiacente ce au proprietăți asemănătoare.

Dacă în interiorul unui obiect nu este ales nici un germene, atunci obiectul respectiv va fi inclus în regiunile ce cresc pornind de la germeni din vecinătatea sa.

10.2 Segmentarea orientată pe contururi

Într-o imagine, variațiile de nivel ale pixelilor reprezintă schimbări ale proprietăților fizice sau geometrice ale obiectelor ce compun scena. Într-un număr mare de cazuri, aceste variații de intensitate corespund frontierelor (conturilor) regiunilor determinate de obiectele dintr-o imagine.

10.2.1 Tehnicile de gradient

Principiul acestor metode constă în definirea punctelor de contur ca fiind acei pixeli ai imaginii pentru care apar schimbări abrupte ale nivelului de gri. Măsurarea acestor variații se face prin operatori derivativi de tip gradient. Derivata imaginii pe direcția r , ce face unghiul θ cu orizontala, este dată de combinația liniară a derivatelor parțiale pe direcțiile orizontală și verticală:

$$\frac{\partial f}{\partial r} = \frac{\partial f}{\partial x} \frac{\partial x}{\partial r} + \frac{\partial f}{\partial y} \frac{\partial y}{\partial r} = \frac{\partial f}{\partial x} \cos\theta + \frac{\partial f}{\partial y} \sin\theta \quad (10.3)$$

$$\frac{\partial f}{\partial r} = f_x \cos\theta + f_y \sin\theta \quad (10.4)$$

Valoarea maximă a acestei derivate, calculate după unghiul θ este determinată de ecuația:

$$\frac{\partial}{\partial \theta} \left(\frac{\partial f}{\partial r} \right) = -f_x \sin\theta + f_y \cos\theta = 0 \quad (10.5)$$

care are soluția:

$$\theta = \arctan \left(\frac{f_x}{f_y} \right) \quad (10.6)$$

Pe această direcție, modulul gradientului este:

$$\left(\frac{\partial f}{\partial r} \right)_{max} = \sqrt{f_x^2 + f_y^2} \quad (10.7)$$

Din punct de vedere practic, implementarea acestei metode implică calcularea, pentru fiecare pixel al imaginii, a derivatelor parțiale f_x și f_y , calcularea modulului gradientului maxim și a direcției acestuia. Valoarea gradientului maxim din fiecare pixel al imaginii este apoi comparată o valoare de prag: dacă pragul este depășit, atunci pixelul este considerat a fi pixel de contur. Realizarea derivatelor parțiale după direcțiile orizontală și verticală implică formularea discretă a lui f_x și f_y :

$$f_x = \frac{\partial f}{\partial x} = \frac{\Delta f(m, n)}{\Delta m} \quad (10.8)$$

$$f_y = \frac{\partial f}{\partial y} = \frac{\Delta f(m, n)}{\Delta n} \quad (10.9)$$

Aceste derivate parțiale discrete pot avea mai multe implementări:

$$f_x = f(m, n) - f(m + 1, n) \quad f_y = f(m, n) - f(m, n + 1) \quad (10.10)$$

$$f_x = f(m-1, n) - f(m, n) \quad f_y = f(m, n-1) - f(m, n) \quad (10.11)$$

$$f_x = f(m-1, n) - f(m+1, n) \quad f_y = f(m, n-1) - f(m, n+1) \quad (10.12)$$

Aceste expresii nu reprezintă altceva decât combinații liniare ale valorilor unor pixeli din imagine, situați în vecinătatea pixelului curent din poziția (m, m) . Prin urmare, se pot implementa folosind următoarele măști de filtrare:

$$W_x = \begin{pmatrix} 1^* & -1 \end{pmatrix} \quad W_y = \begin{pmatrix} 1^* \\ -1 \end{pmatrix}$$

$$W_x = \begin{pmatrix} 1 & -1^* \end{pmatrix} \quad W_y = \begin{pmatrix} 1 \\ -1^* \end{pmatrix}$$

$$W_x = \begin{pmatrix} 1 & 0^* & -1 \end{pmatrix} \quad W_y = \begin{pmatrix} 1 \\ 0^* \\ -1 \end{pmatrix}$$

unde prin $*$ am marcat originea măștii de filtrare. O simplificare uzuală practică este dată de aproximarea:

$$\left(\frac{\partial f}{\partial r} \right)_{max} \approx |f_x| + |f_y| \quad (10.13)$$

Folosirea măștilor de derivare pe verticală și orizontală prezentate are însă serioase neajunsuri: dimensiunea lor mică face ca rezultatele să fie extrem de sensibile la zgomote. În aceste condiții a apărut ideea naturală de a combina filtrarea de derivare cu o filtrare de netezire, care să reducă efectele zgomotului. Ceea ce rezultă pentru operatorii de derivare orizontală și verticală sunt măștile:

$$W_x = \begin{pmatrix} 1 & 0 & -1 \\ c & 0^* & -c \\ 1 & 0 & -1 \end{pmatrix} \quad W_y = \begin{pmatrix} 1 & c & 1 \\ 0 & 0^* & 0 \\ -1 & -c & -1 \end{pmatrix} \quad (10.14)$$

Prin particularizarea valorilor constantei c se pot obtine diverse tipuri de operatori de extragere de contur clasici: Prewitt ($c = 1$), Izotrop ($c = \sqrt{2}$) sau Sobel ($c = 2$). În Figura 10.3 puteți observa efectele extragerii de contur folosind operatorul Sobel.



Figura 10.3: Filtrul Sobel: (a) imaginea originală; (b) imaginea rezultată.

10.2.2 Operatorii compas

Un operator compas este definit de un număr de măști de derivare, corespunzătoare unor filtrări liniare, pe direcțiile principale (verticală, orizontală și cele două diagonale), în cele două sensuri. Compasul clasic are $D = 8$ măști de filtrare, fiecare dintre ele realizând o derivare după o direcție multiplu de 45° . Un exemplu de măști de derivare direcțională sunt măștile următoare, indexate după direcția geografică pe care se calculează derivata:

$$\begin{aligned}
 W_N &= \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} & W_{NV} &= \begin{pmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \\
 W_V &= \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} & W_{SV} &= \begin{pmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{pmatrix} \\
 W_S &= \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix} & W_{SE} &= \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{pmatrix} \\
 W_E &= \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} & W_{NE} &= \begin{pmatrix} 0 & -1 & -1 \\ 1 & 0 & -1 \\ 1 & 1 & 0 \end{pmatrix}
 \end{aligned}$$

Dupa ce se vor calcula, pentru fiecare pixel în parte, cele opt valori ale gradientilor, corespunzătoare celor opt măști de derivare:

$$f_1(m, n) \quad f_2(m, n) \quad \dots \quad f_8(m, n) \quad (10.15)$$

se va determina valoarea maximă dintre aceste opt valori. Această valoare va fi comparată cu valoarea de prag, iar în cazul în care este mai mare, pixelul respectiv va fi considerat pixel de contur.

10.2.3 Identificarea trecerilor prin zero ale celei de-a doua derivate

Unul dintre principalele dezavantaje ale metodelor de gradient este precizia slabă de localizare a conturului în condițiile unei pante puțin abrupte (tranziții slabe, graduale) a acestuia. Derivata a doua poate fi însă folosită pentru a marca centrul tranziției (trecerea sa prin zero). Operatorul bazat pe trecerea prin zero a derivatei secunde este operatorul “zero-crossing”. În cazul imaginilor, derivata secundă trebuie luată în considerare după ambele direcții, combinate în laplacian:

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad (10.16)$$

În continuare sunt prezentate trei măști ce implementează o derivată secundă bidirecțională (operator Laplace):

$$\begin{pmatrix} 0 & -\frac{1}{4} & 0 \\ -\frac{1}{4} & 1 & \frac{1}{4} \\ 0 & \frac{1}{4} & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \end{pmatrix} \begin{pmatrix} -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} \\ -\frac{1}{8} & 1 & -\frac{1}{8} \\ -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} \end{pmatrix}$$

Pentru fiecare pixel din imagine se va calcula derivata a doua și dacă aceasta este zero, atunci pixelul este considerat ca fiind pixel de contur.

DESFĂȘURAREA LUCRĂRII

Problema 1. Implementați segmentarea prin praguirea histogramei. Pentru aceasta veti folosi codul de la lucrarea nr. 4 (egalizarea histogramei) și imaginea `tools.bmp`.

Problema 2. Observați extragerea conturilor prin tehnici de gradient, folosind un operator Prewitt.

```
void ImageViewer :: extragere_contururi( void )
{
    int i, j;
    int k, l;
    int w, h;
```

```

int g, g1, g2;
int h1[ 3 ][ 3 ], h2[ 3 ][ 3 ];

//operatori Prewitt
h1[0][0] = -1;  h1[0][1] = 0;  h1[0][2] = 1;
h1[1][0] = -1;  h1[1][1] = 0;  h1[1][2] = 1;
h1[2][0] = -1;  h1[2][1] = 0;  h1[2][2] = 1;

h2[0][0] = -1;  h2[0][1] = -1; h2[0][2] = -1;
h2[1][0] = 0;   h2[1][1] = 0;   h2[1][2] = 0;
h2[2][0] = 1;   h2[2][1] = 1;   h2[2][2] = 1;

w = image.width();
h = image.height();

QImage image_ext( w, h, 32, 0, QImage::IgnoreEndian );

for( i = 1; i < w - 1; i++ )
    for( j = 1; j < h - 1; j++ )
    {
        g1 = 0;
        g2 = 0;

        for( k = -1; k < 2; k++ )
            for( l = -1; l < 2; l++ )
            {
                g1 += h1[ k + 1 ][ l + 1 ]
                    * qRed( image.pixel( i+k, j+l ) );
                g2 += h2[ k + 1 ][ l + 1 ]
                    * qRed( image.pixel( i+k, j+l ) );
            }

        g = abs( g1 ) + abs( g2 );

        if( g > 100 )
            image_ext.setPixel( i, j, qRgb(255,255,255) );
        else
            image_ext.setPixel( i, j, qRgb(0,0,0) );
    }

image = image_ext;
pm = image;
update();
}

```

Problema 3. Implementați operatorul Izotrop de extragere a contururilor.

Problema 4. Implementați operatorul Sobel de extragere a contururilor.

Problema 5. Implementați operatorul compas dat drept exemplu.

Problema 6. Implementați extragerea de contururi prin identificarea trecerilor prin zero ale derivatei secunde.

Problema 7. Implementați creșterea de regiuni, pornind de la un singur germene. Veți folosi pentru aceasta imaginea `tools.bmp`.

Lista figurilor

2.1	Formarea imaginii în aparatul de fotografiat: 1-lentilă, 2-camera obscură, 3-obiectiv, 4-peliculă.	6
2.2	Forma tipică a funcției de eficiență luminoasă relativă.	7
2.3	Domeniu finit din R^2	8
2.4	Rețea dreptunghiulară de eșantionare.	9
2.5	Exemplu de funcție de cuantizare.	10
3.1	Reprezentarea unei operații punctuale.	16
3.2	Funcția de accentuare de contrast.	16
3.3	Funcția de întindere maximă a contrastului.	17
3.4	Întinderea maximă a contrastului: (a) imaginea originală și (b) imaginea rezultată, pentru $a=50$, $b=200$	18
3.5	Funcția de binarizare.	19
3.6	Binarizarea: (a) imaginea originală și (b) imaginea binarizată cu $T=127$	19
3.7	Funcția de negativare.	20
3.8	Negativarea: (a) imaginea originală și (b) negativul imaginii.	20
3.9	Funcția de decupare cu păstrarea fundalului.	21
3.10	Funcția de decupare fără păstrarea fundalului.	21
4.1	Egalizarea histogramei: (a) imaginea originală și (b) imaginea rezultată.	28
4.2	Histograma (a) originală și (b) după egalizare.	28
5.1	Translatarea unui obiect dreptunghiular.	32
5.2	Translatarea spre dreapta a unui obiect dreptunghiular într-o imagine.	32
5.3	Rotăția.	33
5.4	Oglindirea (a) “stânga-dreapta” (b) “sus-jos”.	33
5.5	Oglindirea “stânga-dreapta”: (a) imaginea originală; (b) imaginea oglindită.	34
6.1	Modelul aditiv de degradare.	38

6.2	Funcția de densitate de probabilitate pentru o distribuție uniformă.	39
6.3	Zgomotul uniform: (a) imaginea originală; (b) imaginea afectată de zgomot uniform, SNR=5 dB.	39
6.4	Funcția de densitate de probabilitate pentru o distribuție gaussiană.	40
6.5	Zgomotul Gaussian: (a) imaginea originală; (b) imaginea afectată de zgomot gaussian, SNR=5 dB.	40
6.6	Zgomotul “sare și piper”: (a) imaginea originală; (b) imaginea cu 10% pixeli afectați de zgomot.	41
6.7	Diverse funcții de distribuție.	42
7.1	Mască de filtrare pătrată de dimensiune 3x3.	48
7.2	Mască de filtrare pătrată de dimensiune 3×3	49
7.3	Mască de filtrare pătrată de dimensiune 5×5	50
7.4	Mască de filtrare pătrată de dimensiune 7×7	50
7.5	Filtrarea de mediere: (a) imaginea originală; (b) imaginea filtrată cu o mască 7x7.	50
7.6	Filtrarea Laplace: (a) imaginea originală; (b) imaginea rezultată (negativată).	52
7.7	Filtrul de accentuare.	53
7.8	Filtrul median: (a) imaginea originală afectată de zgomot “sare și piper”; (b) imaginea filtrată.	54
9.1	Algoritmul Huffman: etapa 1.	71
9.2	Algoritmul Huffman: etapa 2.	71
9.3	Algoritmul Huffman: etapa 3.	72
9.4	Algoritmul Huffman: etapa 4.	72
9.5	Transformarea imaginii într-un șir unidimensional, prin concatenarea liniilor.	73
9.6	Transformarea unei imagini cu 256 nivele de gri, în 8 imagini binare.	74
9.7	Transformarea directă.	75
9.8	Anularea coeficienților de energie mică.	75
9.9	Transformarea inversă.	75
10.1	Exemplu teoretic de segmentare.	82
10.2	Histogramă: (a) bimodală; (b) cu trei moduri dominante. . .	83
10.3	Filtrul Sobel: (a) imaginea originală; (b) imaginea rezultată. .	86

Bibliografie

- [1] **V. Buzuloiu**, “*Prelucrarea Imaginilor*”, note de curs.
- [2] **C. Vertan**, “*Prelucrarea și Analiza Imaginilor*”, Ed. Printech, 1999.
- [3] **A. Jain**, “*Fundamentals of Digital Image Processing*”, Prentice-Hall, 1989.
- [4] **R. Gonzales, R. Woods**, “*Digital Image Processing*”, Addison Wesley, 1992.
- [5] **Trolltech**, “*Qt Overview*”, <http://www.trolltech.com/>
- [6] **Trolltech**, “*The Qt Class Reference*”, <http://www.trolltech.com/>

ISBN (10) 973-635-674-4
ISBN (13) 978-973-635-674-2